METHOD AND APPARATUS FOR STORING PACKETS WITH A PACKET BOUNDARY INDICATOR

FIELD OF THE INVENTION

The present invention is related to the storing of packets along cache lines of a memory with packet boundary indicators. More specifically, the present invention is related to the storing of a plurality of packets along cache lines of a memory with only one packet boundary indicator associated with the plurality of packets so less of the memory is used for the marking function.

BACKGROUND OF THE INVENTION

In the Wide TDM scheme used in the Memory Controller, multiple packets are stored within one data word. Although packet length is contained within the packet data, the Memory Controller does not have enough information to derive packet boundaries. Packet data is striped to multiple Memory Controllers, so no single Memory Controller has enough information to determine the actual length. Storing additional length or packet boundary information for every packet would be quite costly. The Memory Controller can only switch dequeuing priorities at known packet boundaries, so it does need to store some length or packet boundary information.

SUMMARY OF THE INVENTION

The present invention pertains to an apparatus for storing packets. The apparatus comprises a memory for holding packets. The apparatus comprises a mechanism for storing a packet, and preferably at least two packets in the memory with only one packet boundary indicator associated with them.

The present invention pertains to a method for storing packets. The method comprises the steps of receiving packets at a memory. Then there is the step of storing with a memory controller at least two packets in the memory with only one packet boundary indicator associated with them.

BRIEF DESCRIPTION OF THE DRAWINGS

In the accompanying drawings, the preferred embodiment of the invention and preferred methods of practicing the invention are illustrated in which:

Figure 1 is a schematic representation of packet striping in the switch of the present invention.

Figure 2 is a schematic representation of an 0C 48 port card.

Figure 3 is a schematic representation of a concatenated network blade.

Figure 4 is a schematic representation regarding the connectivity of the fabric ASICs.

Figure 5 is a schematic representation of a 32-bit cell transfer.

Figure 6 is a schematic representation regarding back-pressure.

Figure 7 is a schematic representation of a 32-bit packet transferred using external connection number bus.

Figure 8 is a schematic representation of a 64-bit cell transferred.

Figure 9 is a schematic representation of a 64-bit packet transfer.

Figure 10 is a schematic representation of ATM cell flow in the switch.

Figure 11 is a schematic representation of sync pulse distribution.

Figure 12 is a schematic representation regarding the write cycle.

Figure 13 is a schematic representation of the read cycle.

Figure 14 is a schematic representation of the striper ASIC architecture.

Figure 15 is a schematic presentation of the aggregator ASIC architecture.

Figure 16 is a schematic representation of a memory controller ASIC architecture.

Figure 17 is a schematic representation of the wide cache line shared memory architecture.

Figure 18 is a schematic representation of a separator ASIC architecture.

Figure 19 is a schematic representation of an unstriper ASIC architecture.

Figure 20 is a schematic representation regarding the relationship between transmit and receive sequence counters for the separator and unstriper, respectively.

Figure 21 is a schematic representation of a receive synchronizer.

Figure 22 is a schematic representation of an apparatus of the present invention.

Figure 23 is a schematic representation of a memory.

DETAILED DESCRIPTION

Referring now to the drawings wherein like reference numerals refer to similar or identical parts throughout the several views, and more specifically to figures 22 and 23 thereof, there is shown an apparatus 10 for storing packets 12. The apparatus 10 comprises a memory 14 for holding packets 12. The apparatus 10 comprises a mechanism for storing a packet 12, and at least two packets 12 in the memory 14 with only one packet boundary indicator 18 associated with them.

Preferably, the storing mechanism 16 includes a memory controller 20. The memory controller 20 preferably places a packet boundary indicator 18 in the memory 14 after a predetermined number of bits have been stored in the memory 14. Preferably, the memory controller 20 inserts a packet boundary indicator 18 after 15 cache lines worth of packets 12 have been stored in the memory 14. The memory controller 20 preferably switches which packets 12 are to be

transferred from the memory 14 based on packet boundary indicators 18 with respect to priority of the packets 12.

Preferably, the memory 14 is defined by cache lines 22 and the packets 12 are stored along cache lines 22 in the memory 14. The memory controller 20 preferably stores bits of data of packets 12 in a cache line in the memory 14, and there is an identifier 24 in the cache line of data indicating how many bits in the cache line are valid. Preferably, each cache line is 200 bits long. The identifier 24 preferably is 2 bits of the 200 bits of each cache line.

The present invention pertains to a method for storing packets 12. The method comprises the steps of receiving packets 12 at a memory 14. Then there is the step of storing with a memory controller 20 a packet and preferably at least two packets 12 in the memory 14 with only one packet boundary indicator 18 associated with them.

Preferably, the storing step includes the step of placing with the memory controller 20 a packet boundary indicator 18 in the memory 14 after a predetermined number of bits from packets 12 have been stored in the memory 14. The storing step preferably includes the step of storing the packets 12 along cache lines 22 in the memory 14.

Preferably, the placing step includes the step of storing bits of data of packets 12 in a cache line in the memory 14, and an identifier 24 in the cache line of data indicating how many bits in the cache line are valid. After the storing step, there is preferably the step of switching packets 12 from the memory 14 based on packet boundary indicators 18 with respect to priority of the packets 12.

In the operation of the preferred embodiment, the apparatus 10 is used to maintain the minimum amount of packet boundary. In its narrowest sense, it can be thought of as a lossy compression algorithm used to compress packet boundary information. Each cache line of data is stored along with a two-bit field which contains the compressed packet boundary information. A completely uncompressed version of the data would be a single control bit for every data bit indicating that the corresponding data bit was the last bit of a packet. Compared to that form of information, the compression version is 1/100 the size. The apparatus 10 reduces storage space by eliminating unnecessary information.

Each cache line of data is accompanied by a two-bit field indicating how many bits are valid. The values encoded are:

- All 200 bits are valid but do not end on a packet boundary
- All 200 bits are valid and end on a packet boundary
- At least 191 bits are valid and the first non-valid bit is the first 0 found when searching down from 200, and
- Less than 191 bits are valid, with bits 200-193 containing a count of the number of valid bits.

Or equivalently,

- No known packet boundaries
- Packet boundary at 200 bits
- Packet boundary next to the most significant 0, and
- Packet boundary at the bit indicated by data bits 200-193.

By adding only 2 bits per 200 bit entry, it is possible to preserve enough packet boundary information as is necessary by this particular application.

Normally, all 200-bit caches are completely filled. This is efficient, but would cause the loss of all known packet

boundaries. To guarantee that known packet boundaries occur regularly, the logic that fills the caches forces one of the known packet boundary indicators 18 after 15 "no known packet boundary" indicators. The memory 14 is filled in this way along cache lines 22.

When the data in the memory 14 is dequeued, all the data along the cache lines 22 up until a packet boundary indicator 18 is reached is sent out. At the packet boundary indicator 18, the opportunity presents itself for priorities associated with the data to be switched so that if desired, low priority data does not block high priority data.

The general queueing and dequeueing of the packets 12 themselves is well known in the art. The standard storage of packets 12 in the memory 14 is modified by the description herein to minimize the inefficiency of having to store packet boundary indicators 18 at every packet verses the frequency of switching for priority purposes. See U.S. patent application 09/293,563 titled "Very Wide Memory TDM Switching System", incorporated by reference herein, for a more complete description of an overall method and system regarding the transfer of packets 12 that the apparatus and method described herein can be used to advantage.

The switch uses RAID techniques to increase overall switch bandwidth while minimizing individual fabric bandwidth. In the switch architecture, all data is distributed evenly across all fabrics so the switch adds bandwidth by adding fabrics and the fabric need not increase its bandwidth capacity as the switch increases bandwidth capacity.

Each fabric provides 40G of switching bandwidth and the system supports 1, 2, 3, 4, 6, or 12 fabrics, exclusive of the

redundant/spare fabric. In other words, the switch can be a 40G, 80G, 120G, 160G, 240G, or 480G switch depending on how many fabrics are installed.

A portcard provides 10G of port bandwidth. For every 4 portcards, there needs to be 1 fabric. The switch architecture does not support arbitrary installations of portcards and fabrics.

The fabric ASICs support both cells and packets. As a whole, the switch takes a "receiver make right" approach where the egress path on ATM blades must segment frames to cells and the egress path on frame blades must perform reassembly of cells into packets.

There are currently eight switch ASICs that are used in the switch:

- Striper - The Striper resides on the portcard and SCP-IM. It formats the data into a 12 bit data stream, appends a checkword, splits the data stream across the N, non-spare fabrics in the system, generates a parity stripe of width equal to the stripes going to the other fabric, and sends the N+1 data streams out to the backplane.

- Unstriper - The Unstriper is the other portcard ASIC in the the switch architecture. It receives data stripes from all the fabrics in the system. It then reconstructs the original data stream using the checkword and parity stripe to perform error detection and correction.

- Aggregator - The Aggregator takes the data streams and routewords from the Stripers and multiplexes them into a single input stream to the Memory Controller.

- Memory Controller - The Memory controller implements the queueing and dequeueing mechanisms of the switch. This includes the proprietary wide memory interface to achieve the simultaneous en-/de-queueing of multiple cells of data per clock cycle. The dequeueing side of the Memory Controller runs at 80Gbps compared to 40Gbps in order to make the bulk of the queueing and shaping of connections occur on the portcards.

- Separator - The Separator implements the inverse operation of the Aggregator. The data stream from the Memory Controller is demultiplexed into multiple streams of data and forwarded to the appropriate Unstriper ASIC. Included in the interface to the Unstriper is a queue and flow control handshaking.

- Trident - Trident is, strictly speaking, not one of the ASICs. It is actually one-half of the Poseidon chipset. Trident will be used to implement the ATM portcards within the switch.

- Vortex - Vortex is the partner to Trident in the Poseidon chipset. Vortex is the ingress ASIC and Trident the egress device. Together, the two chips implement a 2.5Gbps ingress, 5Gbps egress system capable of supporting up to OC-48c ports.

- Reassembler - The Reassembler ASIC is the frame blade equivalent to Trident. It will be capable of taking cell streams from the Unstriper and converting them into frames.

There are 3 different views one can take of the connections between the fabric: physical, logical, and "active." Physically, the connections between the portcards and the fabrics are all gigabit speed differential pair serial links. This is strictly an implementation issue to reduce the number of signals going over the backplane. The "active" perspective looks at a single switch configuration, or it may be thought of as a snapshot of how data is being processed at a given moment. The interface between the fabric ASIC on the portcards and the fabrics is effectively 12 bits wide. Those 12 bits are evenly distributed ("striped") across 1, 2, 3, 4, 6, or 12 fabrics based on how the fabric ASICs are configured. The "active" perspective refers to the number of bits being processed by each fabric in the current configuration which is exactly 12 divided by the number of fabrics.

The logical perspective can be viewed as the union or max function of all the possible active configurations. Fabric slot #1 can, depending on configuration, be processing 12, 6, 4, 3, 2, or 1 bits of the data from a single Striper and is therefore drawn with a 12 bit bus. In contrast, fabric slot #3 can only be used to process 4, 3, 2, or 1 bits from a single Striper and is therefore drawn with a 4 bit bus.

Unlike previous switches, the switch really doesn't have a concept of a software controllable fabric redundancy mode. The fabric ASICs implement N+1 redundancy without any intervention as long as the spare fabric is installed.

As far as what does it provide; N+1 redundancy means that the hardware will automatically detect and correct a single failure without the loss of any data.

The way the redundancy works is fairly simple, but to make it even simpler to understand a specific case of a 120G switch is used which has 3 fabrics (A, B, and C) plus a spare (S). The Striper takes the 12 bit bus and first generates a checkword which gets appended to the data unit (cell or frame). The data unit and checkword are then split into a 4-bit-per-clock-cycle data stripe for each of the A, B, and C fabrics ($A_3A_2A_1A_0$, $B_3B_2B_1B_0$, and $C_3C_2C_1C_0$). These stripes are then used to produce the stripe for the spare fabric $S_3S_2S_1S_0$ where $S_n = A_n$ XOR B_n XOR C_n and these 4 stripes are sent to their corresponding fabrics. On the other side of the fabrics, the Unstriper receives 4 4-bit stripes from A, B, C, and S. All possible combinations of 3 fabrics (ABC, ABS, ASC, and SBC) are then used to reconstruct a "tentative" 12-bit data stream. A checkword is then calculated for each of the 4 tentative streams and the calculated checkword compared to the checkword at the end of the data unit. If no error occurred in transit, then all 4 streams will have checkword matches and the ABC stream will be forwarded to the Unstriper output. If a (single) error occurred, only one checkword match will exist and the stream with the match will be forwarded off chip and the Unstriper will identify the faulty fabric stripe.

For different switch configurations, i.e. 1, 2, 4, 6, or 12 fabrics, the algorithm is the same but the stripe width changes.

If 2 fabrics fail, all data running through the switch will almost certainly be corrupted.

There are basically two options, both requiring that the defective fabrics be known through some means. Unfortunately, in a double failure system, the hardware that detects and identifies a failed fabric will only be able to identify the fabric that failed first (if there was one). Identifying both the failed fabrics may only be possible through a trial-and-error approach unless the switch software and/or switch diagnostics can develop tests to identify the second failure.

The recommended approach would be to shut down the switch and install as many good fabrics as possible beginning with slot 1. This allows the maximum bandwidth and redundancy be available given the functional hardware available.

The other option is to have the switch software reconfigure the switch to use fewer fabrics. This is an inferior solution for two reasons:

1. It can never provide more bandwidth than the recommended approach.
2. It requires substantial thought and understanding of the switch by the user in order to determine what is the maximum operational configuration.

Basically, the user must start at fabric slot 1 and count the number of operational fabrics. If the spare fabric is operational, then it may be used to "cover" for the first non-operational fabrics.

Example #1: A redundant 240G switch (6+1 fabrics) has suffered fabric failures in slots 3 and 4. Starting with slot 1 there are 2 operational fabrics and the spare is available to cover for slot 3.

This switch can be reconfigured to a 120G non-redundant switch or an 80G redundant switch. Note than by swapping fabric 5 and 6 into slots 3 and 4, this switch could be a 160G redundant switch.

Example #2: A redundant 480G switch suffers fabric failures in slots 1 and the spare. Start swapping fabrics. Slot 1 is dead and the spare is not available to cover for it. This is the worst case scenario.

Example #3: A redundant 480G switch suffers fabric failures in slots 2 and 10. There is one functional fabric counting from slot 1 or 9 if the spare is used to cover for slot 2. This switch can be configured either as 40G redundant or 240G non-redundant. Note that fabrics 7,8, and 9 do not help since the only legal configuration after 6 fabrics is all 12.

The fabric slots are numbered and must be populated in ascending order. Also, the spare fabric is a specific slot so populating fabric slots 1, 2, 3, and 4 is different than populating fabric slots 1, 2, 3, and the spare. The former is a 160G switch without redundancy and the latter is 120G with redundancy.

Firstly, the ASICs are constructed and the backplane connected such that the use of a certain portcard slots requires there to be at least a certain minimum number of fabrics installed, not including the spare. This relationship is shown in Table 0.

In addition, the APS redundancy within the switch is limited to specifically paired portcards. Portcards 1 and 2 are paired, 3 and 4 are paired, and so on through portcards 47 and 48. This means that if APS redundancy is required, the paired slots must be populated together.

To give a simple example, take a configuration with 2 portcards and only 1 fabric. If the user does not want to use APS redundancy, then the 2 portcards can be installed in any two of portcard slots 1 through 4. If APS redundancy is desired, then the two portcards must be installed either in slots 1 and 2 or slots 3 and 4.

Portcard Slot	Minimum # of Fabrics
1-4	1
5-8	2
9-12	3
13-16	4
17-24	6
25-48	12

Table 0: Fabric Requirements for Portcard Slot Usage

To add capacity, add the new fabric(s), wait for the switch to recognize the change and reconfigure the system to stripe across the new number of fabrics. Install the new portcards.

Note that it is not technically necessary to have the full 4 portcards per fabric. The switch will work properly with 3 fabrics installed and a single portcard in slot 12. This isn't cost efficient but it will work.

To remove capacity, reverse the adding capacity procedure.

If the switch is oversubscribed, i.e. install 8 portcards and only one fabric.

It should only come about as the result of improperly upgrading the switch or a system failure of some sort. The reality is that one of two things will occur, depending on how this situation arises. If the switch is configured as a 40G switch and the portcards are added before the fabric, then the 5th through 8th portcards will be dead. If the switch is configured as 80G non-redundant switch and the second fabric fails or is removed then all data through the switch will be corrupted (assuming the spare fabric is not installed). And just to be complete, if 8 portcards were installed in an 80G redundant switch and the second fabric failed or was removed, then the switch would continue to operate normally with the spare covering for the failed/removed fabric.

The switch includes the following features:

- Scales from 40Gbps to 480Gbps (40, 80, 120, 160, 240, 480 GB/sec are the supported configurations).

- Switches ATM cells and variable-length packets

- N+1 fabric redundancy with error detection and recovery supported in the ASIC chipset.

- Native APS support

- Support up to 196K cell shared memory, 9216K unicast and 64K multicast connections.

- Support 2x port speed for fabric dequeueing (2.5 GB/sec in, 5 GB/sec out for each OC48 port).

- Supports both OC48c ports and OC192c ports.

- Provides port/priority queuing similar to past switch fabrics. Four priorities are provided for 40-120 GB/sec switches, 2 priorities/port for 240 GB/sec switches and 1 priority for 480 GB/sec switches.

- ASICs utilize 250 MHz HSTL point to point busses between fabric ASICs and interface with the backplane using standard GBit transceivers.

- Interface to port cards chips use 80-125 MHz LVTTL signals.

- Support output port supplied back-pressure.

The significant architectural difference between the switch and past switches is that incoming traffic is routed to multiple switch fabrics. Each fabric is designed to enqueue 40 GB/sec of data and dequeue 80 GB/sec of data. As data comes into the switch, it is broken up on a bit by bit basis and part of each packet is sent to each fabric in the box. The fabrics will all make the same enqueuing and drop decisions, and all schedule fragments of a packet/cell at the same time. Each fabric sends its portion of the packet or cell to the output port card which reassembles the fragment into the complete cell/packet which is then passed to a shared memory ASIC for per port storage and scheduling. The XOR of the data sent to each fabric is sent to a spare fabric. In the event of a fabric failure, that fabrics data can be recovered by utilizing the good data bits and the parity fabric bits to recalculate any fabrics data. The striping of data to fabrics happens on the basis of 48 bit chunks. This allows the switch to support 1,2,3,4,6 and 12 fabrics.

Five ASICs build the switching functionality for the switch. These ASICs are described briefly below.

TABLE 1: The switch ASICs

ASIC	Function
Striper	Takes incoming cell from Vortex (or OC192c equivalent) or from POS input stage and breaks the data up into the appropriate chunks to go to each fabric, calculates the parity for the spare fabric, concatenates a checksum onto the packet, separates the routeword and data into separate routeword and data busses which run across the backplane.
Aggregator	Receives separate data and routeword busses from multiple stripers. Converts from the reasonably slim dedicated striper->Aggregator busses to a wide shared bus to the memory controllers.
Memory Controllers	Actually perform the queueing of data for the fabrics. Queues the cell into one of 200 queues (192 UC queues, 4 MC queues and 4 control port queues). All drops which occur in the chipset occur here.
Separator	Combines traffic from multiple memory controllers to one fabric output. Provides rate control of the stream of data leaving the fabric for each OC48 or OC192c port.
Unstriper	Receives data from multiple separators. Combines traffic and error checks the received data. Detects errors on any fabric and attempts to reconstruct the good data. Passes the data to the output memory controller. If the striper is on an ATM blade and the data is a packet, it is segmented before passing onto the ATM controller.

Figure 1 shows packet striping in the switch.

The chipset supports ATM and POS port cards in both OC48 and OC192c configurations. OC48 port cards interface to the switching fabrics with four separate OC48 flows. OC192 port cards logically combine the 4 channels into a 10G stream. The ingress side of a port card does not perform traffic conversions for traffic changing between ATM cells and packets. Whichever form of traffic is received is sent to the switch fabrics. The switch fabrics will mix packets and cells and then dequeue a mix of packets and cells to the egress side of a port card.

The egress side of the port is responsible for converting the traffic to the appropriate format for the output port. This convention is referred to in the context of the switch as "receiver makes right". A cell blade is responsible for segmentation of packets and a cell blade is responsible for reassembly of cells into packets. To support fabric speed-up, the egress side of the

port card supports a link bandwidth equal to twice the inbound side of the port card. For each OC48 interface, the unstriper supports a bandwidth of 6GB/sec and for each OC192 interface, a bandwidth of 24 GB/sec (combined routeword + data).

The block diagram for a Poseidon-based ATM port card is shown as in Figure 2. Each 2.5G channel consists of 4 ASICs: Vortex and striper ASIC at the inbound side and unstriper ASIC and Trident ASIC at the outbound side.

At the inbound side, the Vortex ASIC aggregates 1 OC-48c or 4 OC-12c interfaces. Each vortex sends a 2.5G cell stream into a dedicated striper ASIC (using the BIB bus, as described below). The striper converts the vortex supplied routeword into two pieces. A portion of the routeword is passed to the fabric to determine the output port(s) for the cell. The entire routeword is also passed on the data portion of the bus as a routeword for use by the outbound memory controller. The first routeword is termed the "fabric routeword". The routeword for the outbound memory controller is the "egress routeword".

At the outbound side, the unstriper ASIC in each channel takes traffic from each of the port cards, error checks and correct the data and then sends correct packets out on its output bus. The unstriper uses the data from the spare fabric and the checksum inserted by the striper to detect and correct data corruption. The 5Gbps traffic is then sent to the Trident ASIC of the Poseidon chipset. The Trident ASIC stores the incoming cells based on per-VC queues and sends them out to OC- 12c/OC-48c interfaces at aggregated speed of 2.5Gbps.

For the POS interfaces, the striper ASIC input bus speeds up to 3.2Gbps to handle POS overhead. The outbound side, the unstriper talks to a reassembly stage which is currently being defined.

Figure 2 shows an OC48 Port Card.

The OC192 port card supports a single 10G stream to the fabric and between a 10G and 20G egress stream. This board also uses 4 stripers and 4 unstriper, but the 4 chips operate in parallel on a wider data bus. The data sent to each fabric is identical for both OC48 and OC192 ports so data can flow between the port types without needing special conversion functions.

Figure 3 shows a 10G concatenated network blade.

Each 40G switch fabric enqueues up to 40Gbps cells/frames and dequeue them at 80Gbps. This 2X speed-up reduces the amount of traffic buffered at the fabric and lets the outbound ASIC digest bursts of traffic well above line rate. A switch fabric consists of three kinds of ASICs: aggregators, memory controllers, and separators. Nine aggregator ASICs receive 40Gbps of traffic from up to 48 network blades and the control port. The aggregator ASICs combine the fabric route word and payload into a single data stream and TDM between its sources and places the resulting data on a wide output bus. An additional control bus (destid) is used to control how the memory controllers enqueue the data. The data stream from each aggregator ASIC then bit sliced into 12 memory controllers.

The memory controller receives up to 16 cells/frames every 250MHz clock cycle. Each of 12 ASICs stores 1/12 of the

aggregated data streams. It then stores the incoming data based on control information received on the destid bus. Storage of data is simplified in the memory controller to be relatively unaware of packet boundaries (cache line concept). All 12 ASICs dequeue the stored cells simultaneously at aggregated speed of 80Gbps.

Nine separator ASICs perform the reverse function of the aggregator ASICs. Each separator receives data from all 12 memory controllers and decodes the routewords embedded in the data streams by the aggregator to find packet boundaries. Each separator ASIC then sends the data to up to 24 different unstripers depending on the exact destination indicated by the memory controller as data was being passed to the separator.

The dequeue process is back-pressure driven. If back-pressure is applied to the unstriper, that back-pressure is communicated back to the separator. The separator and memory controllers also have a back-pressure mechanism which controls when a memory controller can dequeue traffic to an output port.

In order to support OC48 and OC192 efficiently in the chipset, the 4 OC48 ports from one port card are always routed to the same aggregator and from the same separator (the port connections for the aggregator & Sep are always symmetric.). The table below shows the port connections for the aggregator & sep on each fabric for the switch configurations. Since each aggregator is accepting traffic from 10G of ports, the addition of 40G of switch capacity only adds ports to 4 aggregators. This leads to a differing port connection pattern for the first four aggregators from the second 4 (and also the corresponding separators).

TABLE 2: Agg/Sep port connections

Switch Size	Agg 1	Agg 2	Agg 3	Agg 4	Agg 5	Agg 6	Agg 7	Agg 8
40	1,2,3,4	5,6,7,8	9,10,11,12	13,14,15, 16				
80	1,2,3,4	5,6,7,8	9,10,11,12	13,14,15, 16	17,18,19, 20	21,22,23, 24	25,26,27, 28	29,30,31, 32
120	1,2,3,4	5,6,7,8	9,10,11,12,	13,14,15, 16,	17,18,19, 20	21,22,23, 24	25,26,27, 28	29,30,31, 32
	33,34,35, 36	37,38,39, 40	41,42,43, 44	45,46,47, 48				
160	1,2,3,4	5,6,7,8	9,10,11,12,	13,14,15, 16,	17,18,19, 20,	21,22,23, 24,	25,26,27, 28,	29,30,31, 32,
	33,34,35, 36	37,38,39, 40	41,42,43, 44	45,46,47, 48	49,50,51, 52	53,54,55, 56	57,58,59, 60	61,62,63, 64

Figure 4 shows the connectivity of the fabric ASICs.

The external interfaces of the switches are the Input Bus (BIB) between the striper ASIC and the ingress blade ASIC such as Vortex and the Output Bus (BOB) between the unstriper ASIC and the egress blade ASIC such as Trident.

Two variations of routewords are supported. The first option uses one 32 bit routeword which is passed to the egress board as the egress routeword and has fields extracted to form the fabric routeword. The second option allows the striper to accept both a fabric routeword (which happens on a dedicated routeword bus) and an egress routeword (which is received on the data bus). The second option is more flexible on connection space usage and expansion since that allows all 32 bits of the routeword to be used to identify connections on switch egress.

To maintain compatibility with Vortex, bit 24 is still maintained as the multicast bit. The incoming routeword has the following format.

TABLE 3: 32-bit BIB/BOB route word format

bit 30:25	bit 24	bit 23:0
Connection ID(29:28) & Connection ID(19:16)	Multicast Bit	Connection ID (27:20) & connection ID (15:0)

The 26 bit conn ID in the routeword is set to

MC bit & Connection ID (29:5) for UC connections which are not special routeword values

MC bit & Connection ID (24:0) for MC connections or for special routeword unicast values.

For UC connections, although bits 29:5 are passed to the fabric, only bits 29:20 are used. These bits should be programmed with queue to be used. Bits 29:28 should be programmed with the priority and bits 27:20 programmed with the queue number.

Note that the RW value used for the outbound memory controller is set to

`0' & MC bit & connection ID (29:0).

If the fabric is using 10 bits of conn ID, this leaves 20 bits (1 M connections) for use by the outbound memory controller.

For double routewords, no manipulation is done. The value passed in on the routeword bus needs to equal to the connection ID to be transmitted on the backplane. The following two tables show the routeword value which should be passed on the backplane routeword bus.

TABLE 4: Unicast Connection ID for separate RW bus

bit 25	bit 24:23	Bit 22:15	14:0
Multicast bit=0	Fabric priority	Fabric queue ID	Future expansion bits. This bits are transmitted to the fabric, but the cur rent fabric ignores them. Future fabrics may expand to support these bits.

TABLE 5: Multicast Connection ID for separate RW bus

bit 25	bit 24:23	bit 22:16	bit 15:0
Multicast bit=1	Priority queue ID	Reserved. Note these bits are sent to the fabric to allow future fabrics to support more connection space.	multicast connection ID (0 to 64K) used by the fabric.

Special routewords are flagged by using reserved queue numbers (those in the range of 248-255). These routeword values indicate the receipt of an OAM cell which must get routed to the control port or a queue resynch operation. These special values are always expressed in terms of the connection ID which goes to the fabric. If special routewords are given to the fabric, the memory controller routeword must also be modified if these are getting passed in using the separate connection number bus.

The routeword passed to the fabric will contain the multicast bit and the port mask bits (bits 23:16). The routeword passed to the outbound memory controller will maintain the port mask and also contain the vortex ID and the port ID.

The connection ID of an OAM cell has a special format generated by the Vortex ASIC:

TABLE 6: Connection ID for OAM cell

bit 25	Bit 24:23	bit 22:15	bit 14:9	8	bit 7:0
Multicast bit=0	Vortex ID (7:6)	0xF0 (hex)	Vortex ID (5:0)	reserved	Port ID

The Vortex ID field is used to indicate which source Vortex ASIC the cell comes from. The port ID indicates which port the cell comes from inside the Vortex ASIC. Note that OAM cells are all unicast. All OAM cells are destined to one of 196 blade and control port queues programmed by a 8-bit OAM cell destination register in the memory controller ASICs. If separate routeword busses are being used, bit 24:16 of the BIB_CONN field will be passed to the fabric. The routeword which appears on the data bus (memory controller routeword) should include the port mask, vortex ID and port ID fields in bits 23:0. The value in the multicast bit is a don't care for the memory controller routeword.

Fabric queue ID 0xF0-0xF7 of the unicast connection ID is reserved for software use. All packets which have the fabric queue ID in range of 0xF0-0xFF will be redirected to one of the 4 control port queues based on a programmable register.

The connection ID of a resync cell has the following format. The resync cell is used to resynchronize queues in the memory controller ASICs. Fabric queue ID 0xF8-0xFF of the unicast connection ID is reserved for special fabric functions.

TABLE 7: Connection ID for Resync cell

bit 25	24:23	bit 22:15	bit 14:13	bit 12:0
Multicast bit=0	Priority (unused)	0xFF (hex)	Number of priorities per port	Reserved

The number of priority queues per port can only be changed during the queue resync period, i.e., when a fabric is removed or inserted as follows:

- 00: one priority per port for 480G switch, pick bit 15 down to 8 of the connection ID as the queue ID;
- 01: two priorities per port for 240G switch, pick bit 16 down to 9 of the connection ID as the queue ID;
- 10: 4 priorities per port for 120G or smaller switch, pick bit 17 down to 10 of the connection ID as the queue ID;
- 11: reserved

The resync cell can also be used to copy the shadow data register to a valid location where the shadow address register points to.

Shadow control cell is used to copy the shadow data register to a valid location where the shadow address register points to. The connection ID of a shadow control cell use.

TABLE 8: Connection ID for Shadow Control Cell

bit 24	24:23	bit 22:15	bit 14:0
Multicast bit=0	Priority	0xFE(hex)	Reserved

Data coming into the BIB bus and out of the BOB bus is assumed to be filled onto the busses from most significant bit to least significant bit (highest number bit to lowest number bit).

The Striper ASIC accepts data from the ingress port via the Input Bus (BIB) (also known as DIN_ST_bl_ch bus).

This bus can either operate as 4 separate 32 bit input buses (4xOC48c) or a single 128 bit wide data bus with a common set of control lines to all stripers. This bus supports either cells or packets based on software configuration of the striper chip. It consists of the following signals:

- BIB_Clock: This clock is sourced by the Striper ASIC at up to 100 MHz and is used as a reference for data and control signals on the BIB.

- BIB_BP: This signal is asserted (low) to indicate the striper ASIC cannot take data on the bus due to a bandwidth difference between the BIB and SIB busses. Interfaces which run below 93 MHz will never see this signal asserted. At 100 Mhz, this signal is asserted if more than 65536 bytes of back-to-back data are given. This signal should be sampled at the start of packet. During a packet transfer, this signal will be asserted if the FIFO conditions would cause BP if the packet ended on the current clock cycle. If BP is asserted the clock cycle after the EOP, the striper will effectively ignore the input bus until the BP indication is withdrawn. The packet ingress stage should repeat the first word of the next packet transfer and then proceed with the rest of the packet after the BP signal goes away.

- BIB_Valid_L: This active low input signal delimits valid data on the BIB_SOP, BIB_EOP, and BIB_DATA busses. If this signal is active, the busses are assumed to be valid. If high, the busses are treated as having invalid data for the current clock cycle. If a transfer is not in progress (no SOP without EOP has been given) then the

data bus is treated as invalid even if this signal is a one. For cell interfaces, this signal can be tied active.

- BIB_Cell_Pkt: This signal is set to a one to indicate a cell transfer and a zero to indicate a packet transfer. Signal needs to be valid the same clock cycle as start of cell.

- BIB_Data[127:0]: This is the input 128-bit data bus. If running in 32 bit mode, a cell consists of a 4 byte RW, a 4 byte Header, and twelve 4 byte data words. A packet has a RW and N data words, where $1 \leq N$. If running in 128 bit mode, a cell has a 4 byte RW, a 4 byte header, and 8 bytes of data in the first word, 2 words with 16 bytes of data, and a final word with 8 bytes of data, if the data starts on a word boundary. A following cell can start on the half-word boundary and have all fields offset by 8 bytes. Packets in 128 bit mode work in the same fashion as 32 bit mode, except that EOP and SOP can have larger values. Minimum packet length supported is 16 bytes. If half-word boundary cell starts are used, the correct value (0/4) needs to be given on the SOP bits 3:0.

- BIB_EOP[4:0]: This bus has two fields. Bit 4 is a one to indicate an EOP on the current transfer (if BIB_Valid_L is active). Bit 4 is a zero to indicate no EOP on the current transfer. Bits 3:0 give the offset of the last byte which is valid. The EOP field is not utilized for cell transfers.

- BIB_SOP/C[1:0):This bit indicates a start of packet or cell on the current bus cycle (if BIB_Valid_L is active).

A value of zero indicates start of transfer, a value of one indicates no start of transfer. Asserting bit 1 = 1 indicates that the upper 64 bits carries the SOP and asserting bit 0=1 indicates that the lower 64 bits carries the SOP (for 128 bit bus only). For the 32 bit bus, SOP(0) should be used, SOP(1) should be tied high. For the 128 bit bus, if a packet ends in the upper 64 bits of the bus, a new packet can begin at bit 64.

- BIB_CONN(24:0):This is an optional bus. It can be used to pass a routeword to the striper ASIC to use as the fabric routeword, or the routeword can be transferred as the most significant 32 bits of the first word of data. The data should be valid the same cycle as SOP/C. The value during non-SOP/C cycles is a don't care. The interface is statically configured to either use the separate connection number bus or to expect the routeword on the data bus.

Figure 5 shows a 32 bit BIB cell transfer.

Figure 6 shows a BIB back-pressure.

Figure 7 shows a 32 bit BIB packet transfer using external connection number bus.

The unstriper ASIC sends data to the egress port via Output Bus (BOB) (also known as DOUT_UN_bl_ch bus), which is a 64 (or 256) bit data bus that can support either cell or packet. It consists of the following signals:

This bus can either operate as 4 separate 32 bit output buses (4xOC48c) or a single 128 bit wide data bus with a common set of control lines from all Unstripers. This bus supports either cells or packets based on software configuration of the unstriper chip. It consists of the following signals:

- BOB_Clock: This clock is sourced from the unstriper ASIC at up to 100 MHz and is used as a reference for data and control signals on the BOB.

- BOB_BP: This active low input signal indicates whether data can be transferred (inactive) or cannot be transferred (active). When back-pressure is asserted, the unstriper will stop advancing the output bus and signal data is not valid using the BOB_valid signal. Since synchronization must be done on both sides of the interfaces, 8 clock cycles of data must be allowed from the assertion of BP to data stopping. The source driving BOB_BP cannot make any assumptions on the data stopping or restarting except by examining BOB_Valid.

- BOB_Valid_L: This active low output signal indicates whether the bus has valid data or not during a transfer. This signal indicates invalid data only when BOB_BP has been asserted.

- BOB_Data: This is the output bit data bus. It can either be 64 bits wide or 256 bits wide. If running in 64 bit mode, a cell consists of a word with a 4 byte RW and a 4 byte Header followed by 6 data words. A packet has a RW and N data words, where $1 \leq N$. If running in 256 bit mode and a cell starts on an even 32 byte word boundary, a

cell has a word with a 4 byte RW a 4 byte header and 24 bytes of data in the first word, and a second word with 24 bytes of data. A following cell can start on the next used byte and have all fields offset by 8 bytes. Valid cell start locations are all multiples of 8 (0, 8, 16, 24). Packets in 128 bit mode work in the same fashion as 32 bit mode, except that EOP and SOP can have larger values. Minimum packet length supported is 16 bytes. If half-word boundary cell starts are used, the correct value (0/4) needs to be given on the SOP bits 3:0.

- BOB_EOP: This bit is asserted when the last transfer of a packet is occurring.

- BOB_Cell_Pkt: This signal is set to a one to indicate a cell transfer and a zero to indicate a packet transfer. Signal needs to be valid the same clock cycle as start of cell.

- BOB_SOP/C This bit is a zero to indicate a start of packet or cell on the current bus cycle. Data is always assumed to start at the most significant bit of the bus.

Figure 8 shows a 64 bit BOB cell transfer.

Figure 9 shows a 64 bit BOB packet transfer.

Figure 10 shows an overview of the datapath of the switch ASICs.

The data on the data bus transports an optional byte count (32 bit word, lower 16 bits are the byte count) and a 32 bit egress routeword. The unstriper core will always produce a byte count. If a segmentation engine is used to break the packet up into cells, then the segmentation engine will drop the byte count word before it is given to the cell interface. This dropping is only supported in OC48 mode. In OC192 mode, the chipset will have no provisions for segmentation and dropping the byte count word.

TABLE 9: OC48 BOB format

OC48 Bits	OC192 bits	Label	Usage
63:48	255:240	Unused	reserved for unstriper use
47:32	239:224	Byte count	Gives the count of the number of bytes in the packet not counting the 4 bytes for the egress routeword and the bytes for the byte count (basically, this corresponds to the byte count of the received packet plus/minus any changes for reencapsulation, pushes, or pops.)
31:0	223:192	Egress RW	Routeword for the egress memory controller Next bits start the data (bits (191 to 0) for 192, next clock cycle for OC48

The Synchronizer has two main purposes. The first purpose is to maintain logical cell/packet or datagram ordering across all fabrics. On the fabric ingress interface, datagrams arriving at more than one fabric from one port cards's channels need to be processed in the same order across all fabrics. The Synchronizer's second purpose is to have a port cards's egress channel re-assemble all segments or stripes of a datagram that belong together even though the datagram segments are being sent from more than one fabric and can arrive at the blade's egress inputs at different times. This mechanism needs to be maintained in a system that will have different net delays and varying amounts of clock drift between blades and fabrics.

The switch uses a system of a synchronized windows where start information is transmit around the system. Each transmitter

and receiver can look at relative clock counts from the last resynch indication to synchronize data from multiple sources. The receiver will delay the receipt of data which is the first clock cycle of data in a synch period until a programmable delay after it receives the global synch indication. At this point, all data is considered to have been received simultaneously and fixed ordering is applied. Even though the delays for packet 0 and cell 0 caused them to be seen at the receivers in different orders due to delays through the box, the resulting ordering of both streams at receive time = 1 is the same, Packet 0, Cell 0 based on the physical bus from which they were received.

Multiple cells or packets can be sent in one counter tick. All destinations will order all cells from the first interface before moving onto the second interface and so on. This cell synchronization technique is used on all cell interfaces. Differing resolutions are required on some interfaces.

The Synchronizer consists of two main blocks, mainly, the transmitter and receiver. The transmitter block will reside in the Striper and Separator ASICs and the receiver block will reside in the Aggregator and Unstriper ASICs. The receiver in the Aggregator will handle up to 24(6 port cards x 4 channels) input lanes. The receiver in the Unstriper will handle up to 13(12 fabrics + 1 parity fabric) input lanes.

When a sync pulse is received, the transmitter first calculates the number of clock cycles it is fast (denoted as N clocks).

The transmit synchronizer will interrupt the output stream and transmit N K characters indicating it is locking down.

At the end of the lockdown sequence, the transmitter transmits a K character indicating that valid data will start on the next clock cycle. This next cycle valid indication is used by the receivers to synchronize traffic from all sources. Refer to "K character usage" on page 34 for the mapping of K characters to the functions.

At the next end of transfer, the transmitter will then insert at least one idle on the interface. These idles allow the 10 bit decoders to correctly resynchronize to the 10 bit serial code window if they fall out of synch.

The receive synchronizer receives the global synch pulse and delays the synch pulse by a programmed number (which is programmed based on the maximum amount of transport delay a physical box can have). After delaying the synch pulse, the receiver will then consider the clock cycle immediately after the synch character to be eligible to be received. Data is then received every clock cycle until the next synch character is seen on the input stream. This data is not considered to be eligible for receipt until the delayed global synch pulse is seen.

Since transmitters and receivers will be on different physical boards and clocked by different oscillators, clock speed differences will exist between them. To bound the number of clock cycles between different transmitters and receivers, a global sync pulse is used at the system level to resynchronize all sequence counters. Each chip is programmed to ensure that under all valid clock skews, each transmitter and receiver will think that it is fast by at least one clock cycle. Each chip then waits for the appropriate number of clock cycles they are into their current sync_pulse_window. This ensure that all sources run N* sync_pulse_window valid clock cycles between synch pulses.

As an example, the synch pulse window could be programmed to 100 clocks, and the synch pulses sent out at a nominal rate of a synch pulse every 10,000 clocks. Based on a worst case drifts for both the synch pulse transmitter clocks and the synch pulse receiver clocks, there may actually be 9,995 to 10,005 clocks at the receiver for 10,000 clocks on the synch pulse transmitter. In this case, the synch pulse transmitter would be programmed to send out synch pulses every 10,006 clock cycles. The 10,006 clocks guarantees that all receivers must be in their next window. A receiver with a fast clock may have actually seen 10,012 clocks if the synch pulse transmitter has a slow clock. Since the synch pulse was received 12 clock cycles into the synch pulse window, the chip would delay for 12 clock cycles. Another receiver could seen 10,006 clocks and lock down for 6 clock cycles at the end of the synch pulse window. In both cases, each source ran 10,100 clock cycles.

When a port card or fabric is not present or has just been inserted and either of them is supposed to be driving the inputs of a receive synchronizer, the writing of data to the particular input FIFO will be inhibited since the input clock will not be present or unstable and the status of the data lines will be unknown. When the port card or fabric is inserted, software must come in and enable the input to the byte lane to allow data from that source to be enabled. Writes to the input FIFO will be enabled. It is assumed that, the enable signal will be asserted after the data, routeword and clock from the port card or fabric are stable.

At a system level, there will be a primary and secondary sync pulse transmitter residing on two separate fabrics. There will also be a sync pulse receiver on each fabric and blade. This

can be seen in Figure 11. A primary sync pulse transmitters will be a free-running sync pulse generator and a secondary sync pulse transmitter will synchronize its sync pulse to the primary. The sync pulse receivers will receive both primary and secondary sync pulses and based on an error checking algorithm, will select the correct sync pulse to forward on to the ASICs residing on that board. The sync pulse receiver will guarantee that a sync pulse is only forwarded to the rest of the board if the sync pulse from the sync pulse transmitters falls within its own sequence "0" count. For example, the sync pulse receiver and an Unstriper ASIC will both reside on the same Blade. The sync pulse receiver and the receive synchronizer in the Unstriper will be clocked from the same crystal oscillator, so no clock drift should be present between the clocks used to increment the internal sequence counters. The receive synchronizer will require that the sync pulse it receives will always reside in the "0" count window.

If the sync pulse receiver determines that the primary sync pulse transmitter is out of sync, it will switch over to the secondary sync pulse transmitter source. The secondary sync pulse transmitter will also determine that the primary sync pulse transmitter is out of sync and will start generating its own sync pulse independently of the primary sync pulse transmitter. This is the secondary sync pulse transmitter's primary mode of operation. If the sync pulse receiver determines that the primary sync pulse transmitter has become in sync once again, it will switch to the primary side. The secondary sync pulse transmitter will also determine that the primary sync pulse transmitter has become in sync once again and will switch back to a secondary mode. In the secondary mode, it will sync up its own sync pulse to the primary sync pulse. The sync pulse receiver will have less tolerance in its sync pulse filtering mechanism than the secondary sync pulse

transmitter. The sync pulse receiver will switch over more quickly than the secondary sync pulse transmitter. This is done to ensure that all receiver synchronizers will have switched over to using the secondary sync pulse transmitter source before the secondary sync pulse transmitter switches over to a primary mode.

Figure 11 shows sync pulse distribution.

In order to lockdown the backplane transmission from a fabric by the number of clock cycles indicated in the sync calculation, the entire fabric must effectively freeze for that many clock cycles to ensure that the same enqueuing and dequeueing decisions stay in sync. This requires support in each of the fabric ASICs. Lockdown stops all functionality, including special functions like queue resynch.

The sync signal from the synch pulse receiver is distributed to all ASICs. Each fabric ASIC contains a counter in the core clock domain that counts clock cycles between global sync pulses. After the sync pulse if received, each ASIC calculates the number of clock cycles it is fast.(δ). Because the global sync is not transferred with its own clock, the calculated lockdown cycle value may not be the same for all ASICs on the same fabric. This difference is accounted for by keeping all interface FIFOs at a depth where they can tolerate the maximum skew of lockdown counts.

Lockdown cycles on all chips are always inserted at the same logical point relative to the beginning of the last sequence of "useful" (non-lockdown) cycles. That is, every chip will always execute the same number of "useful" cycles between lockdown events, even though the number of lockdown cycles varies.

Lockdown may occur at different times on different chips. All fabric input FIFOs are initially set up such that lockdown can occur on either side of the FIFO first without the FIFO running dry or overflowing. On each chip-chip interface, there is a sync FIFO to account for lockdown cycles (as well as board trace lengths and clock skews). The transmitter signals lockdown while it is locked down. The receiver does not push during indicated cycles, and does not pop during its own lockdown. The FIFO depth will vary depending on which chip locks down first, but the variation is bounded by the maximum number of lockdown cycles. The number of lockdown cycles a particular chip sees during one global sync period may vary, but they will all have the same number of useful cycles. The total number of lockdown cycles each chip on a particular fabric sees will be the same, within a bounded tolerance.

The Aggregator core clock domain completely stops for the lockdown duration - all flops and memory hold their state. Input FIFOs are allowed to build up. Lockdown bus cycles are inserted in the output queues. Exactly when the core lockdown is executed is dictated by when DOUT_AG bus protocol allows lockdown cycles to be inserted. DOUT_AG lockdown cycles are indicated on the DestID bus.

The memory controller must lockdown all flops for the appropriate number of cycles. To reduce impact to the silicon area in the memory controller, a technique called propagated lockdown is used.

The aggregator signals lockdown cycles on the DIN_ME bus. The memory controller does not push during these cycles. The memory controller does not pop during lockdown to account for the non-push cycles. The FIFO depth is set during fabric

synchronization to tolerate getting deeper or shallower depending on who locks down first.

Lockdown idle cycles are inserted on the DOUT and CH_ID busses. An extended sync signal is used to indicate the number of lockdown cycles on the DOUT_ME bus to aid the Separator's lockdown function.

The token bus lockdown looks the same as the DIN_ME bus from a memory controller perspective. Non-push cycles are signaled by the separators according to their lockdowns. The memory controller does not pop during lockdown. The Separator locks down completely in a manner similar to the Aggregator. DIN_SP and CH_ID lockdown cycles are signaled individually per-bus via the SYNC signals. Any continuous SYNC assertion after the first one is considered a lockdown cycle. Lockdown bus cycles are not pushed into the input FIFOs.

The chip-to-chip communication within a single fabric must be synchronized. Although no clock drift exists between chips, differences in track delays cause data to arrive at different Memory Controllers at different times. All Memory Controllers need to process incoming packets in exactly the same logical order on each chip. The Separators must align and combine multiple data slices coming from different Memory Controllers. The Memory Controllers must take the tokens received from the Separators and apply them at exactly the same point in the logical packet flow, or drop decisions may differ from chip to chip.

The on-fabric chip-to-chip synchronization is executed at every sync pulse. While some sync error detecting capability may exist in some of the ASICs, it is the Unstriper's job to detect

fabric synchronization errors and to remove the offending fabric. The chip-to-chip synchronization is a cascaded function that is done before any packet flow is enabled on the fabric. The synchronization flows from the Aggregator to the Memory Controller, to the Separator, and back to the Memory Controller. After the system reset, the Aggregators wait for the first global sync signal. When received, each Aggregator transmits a local sync command (value 0x2) on the DestID bus to each Memory Controller.

The Memory Controllers do not push anything into a DIN input FIFO until the first sync command is seen on that bus. The sync and every bus cycle following is constantly pushed into the input FIFO. On the core side of the input FIFOs, no FIFO is popped until a sync appears in the FIFO from every Aggregator. After two additional margin cycles, every input FIFO is popped every cycle. After this point the input FIFO depths remain constant. The depths are roughly a function of the track delays from each Aggregator. Immediately after the Memory Controllers begin sampling the Aggregator input FIFOs, a sync signal (S_SYNC_L) is transmitted to all Separators on the DOUT and CH_ID busses.

Like the Memory Controllers, the Separators do not push into the DIN and CH_ID busses until a sync signal is received on that bus. The sync and everything after is constantly pushed into the input FIFO.

On the core side the Separator always waits until at least one word is present on all input busses, and then pops the CH_ID and DIN busses simultaneously. This will logically align the data stripes coming from the Memory Controllers. After the first combined sync is popped from the input FIFOs, the Separators send a sync signal on the TOKEN bus to the Memory Controllers.

The Memory Controllers do not push into the TOKEN bus input FIFO until a sync signal (0x3F on the token bus) has been seen on the bus. The sync and all subsequent tokens and idles are always pushed.

All Memory Controllers need to apply the received tokens to the same point in the incoming logical flow in order for all drop decisions to be identical. This is done by waiting a worst case number of clock cycles after the Separator sync transmission before beginning to pop the token input FIFO. The worst case delay must be used because there is no way for a single Memory Controller to know exactly when all other Memory Controllers have received a token. The programmable delay stored in the 16-bit Token Sync Wait Register is in "useful" cycles (125MHz) that do not include the fabric lockdown cycles. The worst case delay is the worst case skew for all data paths going from the Aggregator to Memory Controller to Separator and back to Memory Controller.

The following Table 10 gives the min/max delays which the chipset supports and represent the limits of what is verified in the chip verification process.

Sync pulse transport delay from Transmitter to any individual chip receiving the sync pulse (WC path - BC path): 500 nS (min delay of 0, max delay of 500 nS). At 175 ps/inch, this works out to a difference of about 70m. Backplane transport delay difference from local sync pulse receipt to reception of the sync indication flag by the far end chips: 500 nS. Note that it is desired to allot about 25 nS of this to the chip synchronizer operation which gives a delta path delay supported of 500 nS.

Oscillators should be 100 ppm oscillators. The assumption of the design was that the difference in transmission path delay was less than or equal to clock drift. On board delays between chips have been designed to exceed the following specs:

Shortest net: 0.25", transport delay of pretty much 0.
Longest net: 25", transport delay is 5 nS.

For any signal distribution. The net delta delay between chips is a multiplier of the number of busses the sync has traversed. Since the sync goes through a receive synchronization to the local clock of the chip, an +/- 8 nS uncertainly has to be added at each stage giving a net uncertainty of around 21 nS for each hop.

TABLE 10: Fabric sync delay

Chip	Number busses	ofSkew	Notes
Agg	1	21 nS	Sync pulse in
Memory controller DIN	2	42 nS	Sync pulse to agg + agg_mc delta
Sep DIN	3	63 nS	Sync pulse to agg + agg_mc + mc_sep (note this sync pulse is delayed by the memory controller for propagated lockdown).
memory controller token_in	4	84nS	Everything above + sep_mc tokens.

The control port follows the same cell flow as the regular ports. The switch control processor sends cells to the striper ASIC; the striper stripes the cells and route words across all fabrics. An additional aggregator (9th) ASIC sends cells via the DOUT_AG/DestID buses to all 12 memory controllers. Each memory controller ASIC has an additional 9th DIN_ME_fb_se_9 bus.

The memory controller ASIC will route the incoming control port cells to any one of the control port destination queues and blade queues (up to 196 queues). The 9th DOUT_ME_fb_se_9 bus is used to send the control cells to the 9th separator ASIC, which sends the cells to one of several destination unstriper ASICs. The unstriper ASIC reconstructs the cells from all 9th separator ASICs across all fabrics. It sends the complete control cells to the switch control processor it is connected to.

Note that the control port destination queues can be part of any multicast cells such that the multicast port mask is necessary to include additional bit(s) to indicate the control port queue(s).

There are at most 4 control ports in any switch configurations. This limitation is due to the aggregator and separator ASICs only have 4 12-bit channels which can be scalable to different switch configurations, respectively. In other words, bus DIN_AG_fb_9_1_1, DIN_AG_fb_9_2_1, DIN_AG_fb_9_3_1, and DIN_AG_fb_9_4_1 of the aggregator ASIC are connected to up to 4 control port striper ASICs. Bus DOUT_SP_fb_9_1_1, DOUT_SP_fb_9_2_1, DOUT_SP_fb_9_3_1, and DOUT_SP_fb_9_4_1 of the separator ASIC are connected to up to 4 control port unstriper ASICs.

The striping function assigns bits from incoming data streams to individual fabrics. Two items were optimized in deriving the striping assignment:

1. Backplane efficiency should be optimized for OC48 and OC192.
2. Backplane interconnection should not be significantly altered for OC192 operation.

These were traded off against additional muxing legs for the striper and unstriper ASICs. Irregardless of the optimization, the switch must have the same data format in the memory controller for both OC48 and OC192.

Backplane efficiency requires that minimal padding be added when forming the backplane busses. Given the 12 bit backplane bus for OC48 and the 48 bit backplane bus for OC192, an optimal assignment requires that the number of unused bits for a transfer to be equal to (number_of_bytes *8)/bus_width where "/" is integer division. For OC48, the bus can have 0, 4 or 8 unutilized bits. For OC192 the bus can have 0, 8, 16, 24, 32, or 40 unutilized bits.

This means that no bit can shift between 12 bit boundaries or else OC48 padding will not be optimal for certain packet lengths.

For OC192c, maximum bandwidth utilization means that each striper must receive the same number of bits (which implies bit interleaving into the stripers). When combined with the same backplane interconnection, this implies that in OC192c, each stripe must have exactly the correct number of bits come from each striper which has 1/4 of the bits.

For the purpose of assigning data bits to fabrics, a 48 bit frame is used. Inside the striper is a FIFO which is written 32 bits wide at 80-100 MHz and read 24 bits wide at 125 MHz. Three 32 bit words will yield four 24 bit words. Each pair of 24 bit words is treated as a 48 bit frame. The assignments between bits and fabrics depends on the number of fabrics.

TABLE 11: Bit striping function

		Fab 0	Fab 1	Fab 2	Fab 3	Fab 4	Fab 5	Fab 6	Fab 7	Fab 8	Fab 9	Fab 10	Fab 11
	0:11	0:11											
1 fab	12:23	12:23											
	24:35	24:35											
	36:47	36:47											
	0:11	0 , 2 , 5 , 7,8,10	1 , 3 , 4 , 6,9,11										
2 fab	12:23	1 3 , 1 5 , 16,18, 21	1 2 , 1 4 , 1 7 , 1 9 , 20,22										
	24:35	+24 to 0:11	+24 to 0:11										
	36:47	+24 to 12:23	+24 to 12:23										
	0:11	0,3,5, 10	2,4,7, 9	1,6,8, 11									
3 fab	12:23	1 5 , 1 7 , 22,13	1 4 , 1 6 , 19,21	1 3 , 1 8 , 20,23									
	24:35	+24 to 0:11	+24 to 0:11	+24 to 0:11									
	36:47	+24 to 12:23	+24 to 12:23	+24 to 12:23									
	0:11	0,5,10	3,4,9	2,7,8	1,6,11								
4fab	12:23	15,16, 21	14,19, 20	13,18, 23	12,17, 22								
	24:35	26,31,32	25,30, 35	2 4 , 2 9 , 34'	27,28, 33								
	36:47	37,42,47	36,41, 46	39,40, 43	38,43, 44								
	0:11	0,11	1,4	5,8	2,9	3,6	7,10						
6 fab	12:23	14,21	15,18	19,22	12,23	13,16	17,20						
	24:35	+24 to 0:11											
	36:47	+24 to 12:23											
	0:11	0	4	8	1	5	9	2	6	10	3	7	11
12 fab	12:23	15	19	23	12	16	20	13	17	21	14	18	22
	24:35	26	30	34	27	31	35	24	28	32	25	29	33
	36:47	37	41	45	38	42	46	39	43	47	37	40	44

The following tables give the byte lanes which are read first in the aggregator and written to first in the separator. The four channels are notated A,B,C,D. The different fabrics have

different read/write order of the channels to allow for all busses to be fully utilized.

One fabric-40G

The next table gives the interface read order for the aggregator.

Fabric	1st	2nd	3rd	4th
0	A	B	C	D
Par	A	B	C	D

Two fabric-80G

Fabric	1st	2nd	3rd	4th
0	A	C	B	D
1	B	D	A	C
Par	A	C	B	D

120G

Fabric	1st	2nd	3rd	4th
0	A	D	B	C
1	C	A	D	B
2	B	C	A	D
Par	A	D	B	C

Three fabric-160G

Fabric	1st	2nd	3rd	4th
0	A	B	C	D
1	D	A	B	C
2	C	D	A	B
3	B	C	D	A
Par	A	B	C	D

Siz fabric-240 G

Fabric	1st	2nd	3rd	4th
0	A	D	C	B
1	B	A	D	C
2	B	A	D	C
3	C	B	A	D
4	D	C	B	A
5	D	C	B	A
Par	A	C	D	B

Twelve Fabric-480 G

Fabric	1st	2nd	3rd	4th
0,1,2	A	D	C	B
3,4,5	B	A	D	C
6,7,8	C	B	A	D
9,10,11	D	C	B	A
Par	A	B	C	D

Interfaces to the gigabit transceivers will utilize the transceiver bus as a split bus with two separate routeword and data busses. The routeword bus will be a fixed size (2 bits for OC48 ingress, 4 bits for OC48 egress, 8 bits for OC192 ingress and 16 bits for OC192 egress), the data bus is a variable sized bus. The transmit order will always have routeword bits at fixed locations. Every striping configuration has one transceiver that it used to talk to a destination in all valid configurations. That transceiver will be used to send both routeword busses and to start sending the data.

The backplane interface is physically implemented using 125 MHz interfaces to the backplane transceivers. The 125 MHz bus for both ingress and egress is viewed as being composed of two halves, each with routeword data. The two bus halves may have information on separate packets if the first bus half ends a packet.

For example, an OC48 interface going to the fabrics locally speaking has 24 data bits and 2 routeword bits @125 MHz. This bus will be utilized acting as if it has 2x (12 bit data bus + 1 bit routeword bus). The two bus halves are referred to as A and B. Bus A is the first data, followed by bus B. A packet can start on either bus A or B and end on either bus A or B.

In mapping data bits and routeword bits to transceiver bits, the bus bits are interleaved. This ensures that all transceivers should have the same valid/invalid status, even if the striping amount changes. Routewords should be interpreted with bus A appearing before bus B.

The bus A/Bus B concept closely corresponds to having 250 MHz interfaces between chips.

All backplane busses support fragmentation of data. The protocol used marks the last transfer (via the final segment bit in the routeword). All transfers which are not final segment need to utilize the entire bus width, even if that is not an even number of bytes. Any given packet must be striped to the same number of fabrics for all transfers of that packet. If the striping amount is updated in the striper during transmission of a packet, it will only update the striping at the beginning of the next packet.

Each transmitter on the ASICs will have the following I/O for each channel:

8 bit data bus, 1 bit clock, 1 bit control.

On the receive side, for channel the ASIC receives

a receive clock, 8 bit data bus, 3 bit status bus.

The switch optimizes the transceivers by mapping a transmitter to between 1 and 3 backplane pairs and each receiver with between 1 and 3 backplane pairs. This allows only enough transmitters to support traffic needed in a configuration to be populated on the board while maintaining a complete set of backplane nets. The motivation for this optimization was to reduce the number of transceivers needed.

The optimization was done while still requiring that at any time, two different striping amounts must be supported in the gigabit transceivers. This allows traffic to be enqueued from a striping data to one fabric and a striper striping data to two fabrics at the same time.

In all modes of operation, the entire 3.0G of data is always supported on switch ingress. For egress operation, for 40G and 80G, the number of transceivers needed to support a full 2x speedup was deemed to expensive. For these switch modes, the output speedup is between 1.5 and 2. All configurations above 80G support a full 2x speedup.

Depending on the bus configuration, multiple channels may need to be concatenated together to form one larger bandwidth pipe (any time there is more than one transceiver in a logical

connection. Although quad gbit transceivers can tie 4 channels together, this functionality is not used. Instead the receiving ASIC is responsible for synchronizing between the channels from one source. This is done in the same context as the generic synchronization algorithm.

The 8b/10b encoding/decoding in the gigabit transceivers allow a number of control events to be sent over the channel. The notation for these control events are K characters and they are numbered based on the encoded 10 bit value. Several of these K characters are used in the chipset. The K characters used and their functions are given in the table below.

TABLE 12: K Character usage

K character	Function	Notes
28.0	Sync indication	Transmitted after lockdown cycles, treated as the prime synchronization event at the receivers
28.1	Lockdown	Transmitted during lockdown cycles on the backplane
28.2	Packet Abort	Transmitted to indicate the card is unable to finish the current packet. Current use is limited to a port card being pulled while transmitting traffic
28.3'	Resync window	Transmitted by the striper at the start of a synch window if a resynch will be contained in the current sync window
28.4	BP set	Transmitted by the striper if the bus is currently idle and the value of the bp bit must be set.
28.5	Idle	Indicates idle condition
28.6	BP clr	Transmitted by the striper if the bus is currently idle and the bp bit must be cleared.

The switch has a variable number of data bits supported to each backplane channel depending on the striping configuration for a packet. Within a set of transceivers, data is filled in the following order:

F[fabric]_[oc192 port number][oc48 port designation (a,b,c,d)][transceiver_number]

Everything in the documentation is done for fabric=1, which is the case where all connections are needed. The only part of this which is used for fill order is transceiver_number (OC48) and transceiver number and oc48 port designation for OC192.

The fundamental rules for mapping are the following:

1. BP + RW are on transceiver 1 These always occupy the first 4 bits of the transceiver.

2. Data bits starting with the least significant bit are filled into the data bus in a 2 bit bit-interleaved pattern, with bus A and bus B pairs.

3. Transceivers are filled in starting at bit 0 of their transmit and receive interfaces.

4. All multibit routeword fields are transmitted LSB to MSB. This includes connection number, number of fabrics and encoded values of stop/align/final segment. The overall routeword is notated as starting from bit 0 (least significant bit) and up. Transmit order is Bit 0 (SOP) goes on the first routeword bit, followed by bit 1 (Packet type). If multiple routeword bits are transmitted in the same clock they are filled in starting with the first bit going to bit 0, the second bit going to bit 1.

5. Data should be encoded and decoded based on a bus A/Bus B order.

6. For OC192, the fill order should be bus A, B, C, D for routeword bits. For data bits, the fill order depends on wacking/unwacking/reverse unwacking and reverse wacking functions.

Transceiver 1

For an ingress bus, the format of data is the following:

Bit 0- BP
Bit 1- 0
Bit 2- RWA
Bit 3- RWB
Bit 4-Dataa(0)
Bit 5-Dataa(1)
Bit 6 Datab(0)
Bit 7 Datab(1)

Note that for 12 fabric mode, bits 5 and 7 are unused. The location of datab(0) does not change.

For the egress bus, the format of the data is the following:

Bit 0- RWA(0)
Bit 1- RWA(1)
Bit 2- RWB(0)
Bit 3- RWB(1)
Bit 4-Dataa(0)
Bit 5-Dataa(1)
Bit 6 Datab(0)
Bit 7 Datab(1)

Transceiver 2 and up

Fill up the data bus starting at each transceiver bit 0 to bit 7 with 2 bit interleaved

dataa/datab patterns.

For example, transceiver 2 has the following pattern:

Bit 0- dataa(2)

Bit 1- dataa(3)

Bit 2- datab(2)

Bit 3- datab(3)

Bit 4-Dataa(4)

Bit 5-Dataa(5)

Bit 6 Datab(4)

Bit 7 Datab(5)

The stop/align encoding depends on the width of the bus interface.

TABLE 13: OC48 portcard to fabric routeword stop/align

Field	Length	Function
Stop/Align /FS	2 + n (where n is the number of clock cycles of transfer)	In this mode, this field is stop & align & final_segment. Stop bit is a 1 to indicate no stop, zero indicates stop. Stop bits repeat in a serial stream until a stop bit of zero is seen, followed by the align bit and FS. Since stop is followed by the align and FS bits, the stop bit is given 2 clock cycles before the end of data. Align bit is a one to indicate valid data on the last complete byte on the interface. For odd 12 bit words(assuming zero based counting), align = 0 indicates bits 0:3 are valid, and bits 4:11 are invalid. Align = 1 for these words indicates that all 12 bits are valid. For even words, align should normally be a 1. Short packets are indicated by signaling a stop on byte 53 of the transfer. In reality, 54 bytes will be transferred, but the packet is flagged as a short packet. Final segment is a one to indicate a final segment of a packet and a zero to indicate a partial segment of a packet. Only one packet can be in transit at any one time on this bus. This bit is only valid for packets. For cells this bit should be a one. Packets which are not final segments should be terminated only on odd cycles with all bits utilized.

TABLE 14: OC192 portcard to fabric routeword stop/align

Field	Length	Function
Stop/Align /FS	3 + 4 * number of extra clocks	Due to length restrictions on this bus, the stop/align has to be treated differently than for OC48 transfers. The first clock cycle, this field is 3 bits long and is notated as SAF0. In all future clock cycles the stop field is 4 bits long and notated SAF1. The definitions of SAF0 and SAF1 are given below. SAF0(0). Bit zero is a zero to indicate a stop, a one to indicate no stop. SAF0(2:1)-"00" indicates full word transfer. "01" indicates a full word transfer but for a short packet. "10" indicates a full word transfer but not the final segment. "11" is reserved. SAF1(0) Bit zero is a zero to indicate a stop, a one to indicate no stop on the current cycle. SAF1(3:1)-binary value of the number of valid bytes. Zero is reserved and 7 is used to indicate 6 bytes valid but not the final segment. 6 indicates 6 bytes valid and final segment. All partial word transfers automatically indicate an implied final segment.

TABLE 15: OC48 Fabric-Port card routeword stop/align

Field	Length	Function
Stop/Align /FS	3 + 2 * number of extra clocks	Value is treated as a repeated 2 bit value (encoded stop) followed by the final segment bit. Stop field is interpreted as: 11-continue 00-1st byte finished is valid and stop 01-2nd bytes finished is valid and stop 10-3rd byte finished is valid and stop, or non-final segment. Short packets are indicated by flagging a stop at byte 53. Final segment is a one for a final segment, a zero for a continuing packet. For final segments, the stop field should be encoded as a "10"

The port card - fabric interface at OC192 variable routeword bits are given in the table below.

TABLE 16: OC192 Fabric-port card routeword stop/align

Field	Length	Function
Stop/Align	7 + 8* number of extra clock cycles of	Bit 0 indicates stop. Zero indicates stop, 1 continue. Bits 4:1 give the number of valid bytes which complete on the interface if a stop is being executed. If no stop is being executed the value of these bits are don't cares. Zero is reserved. 0xC indicates 12 bytes and final segment. 0xD indicates the full bus and packet continues (not a final segment). Values 0xE, 0xF are reserved. Any non-12 byte ending offset automatically signals end of segment.

	transfer	Bit 5:6 (first cycle) and bits 5:7 (second cycle and on) are reserved. The purpose of these bits is to align the next stop field with the following clock cycle of data. Short packets are indicated by flagging a stop at byte 53.

Depending on the switch configuration, the bus may not transfer an integer number of bytes. This is handled by the interface always flagging the bytes which finish and the transmit and receive state machines must track where bytes begin and end based on the current cycle in the transfer.

The bus consists of a multiplexed address/data bus (AD_DATA), a select signal (AD_SEL_L), a read/write signal (AD_RW), and a bus transaction complete indication signal (AD_RDY_L). AD bus is used for read/write access of control/status registers.

In order to write to a control/status register, the read/write signal (AD_RW) must be low. The select signal (AD_SEL_L) must be asserted low for the entire duration of the access, and values must be placed on the AD_DATA bus in the following sequence (cycle 0 is the first cycle where AD_SEL_L is low for this transaction):

- cycle 2-5: Data to be written to control/status register. For registers that are wider than 8-bits (maximum of 32-bits) write data must be presented one byte per cycle starting with LSB. Any data presented on the bus beyond the width of the register will be ignored.
- cycles > 5: ASIC will assert AD_RDY_L on completion of the write access, and will keep it asserted until AD_SEL_L is de-asserted.

Figure 12 shows a Write Cycle.

In order to read from a control/status register, the read/write signal (AD_RW) must be high. The select signal (AD_SEL_L) must be asserted low for the entire duration of the access, and values must be placed on the AD_DATA bus in the following sequence (cycle 0 is the first cycle where AD_SEL_L is low for this transaction):

- cycle 0-1: Address of control/status register
- cycle 2: AD_DATA bus should be released (hi-z)
- cycles >3: When the data is available, ASIC will drive the read data onto the bus, one byte per cycle for four cycles, along with assertion of AD_RDY_L signal. For registers smaller than 32-bits wide, unused bits are presented as zeros. The LSB is present on the bus during the 1st clock cycle of AD_RDY_L assertion.

Figure 13 shows a Read Cycle.

The switch chips will generate interrupts on error conditions. The interrupt lines have the following characteristics:

1. Level Sensitive

2. Active Low

3. Asynchronous (no clock generated to go along with the interrupt).

4. Assume point-to-point interconnection with board logic which combines together interrupts.

Interrupts are maskable on a condition by condition basis inside each chip. The interrupt signal is asserted on the occurrence of an error condition and is cleared when the error condition is cleared. Any temporary conditions which caused an interrupt are recorded in the chip so no phantom interrupts should be seen.

The reality of the switch is that errors will occur. The intent in the following is to detail the expected system behavior and recovery strategy needed for each error type.

TABLE 17: Error recovery in the ASICs

Error	Detection Mechanism	Error recovery required	Hardware comments
Stuck bit on port card egress	unstriper sees data corruption from one fabric		
Stuck bit between agg & memory controller	unstriper sees data corruption from one fabric, either route word or data.		
Stuck bit between memory controller & separator	unstriper sees data corruption from one fabric, either route word or data		
Stuck bit on fabric egress			
Soft-fail on routeword from port card	At least two unstripers see either a routeword mismatch, a state with a high number of routeword mismatches, or data parity errors or any number of unstripers will see a routeword mismatch, a high number of routeword mis matches or data parity errors and an aggregator will see a synch error.	Queue resynch	Worst case scenario involves failing routeword with different fabric routewords to fabrics. Either queueing a packet to the wrong port or dropping the traffic in the aggregator can cause an impact to all ports. Probability of impacting more ports goes up with traffic load and memory utilization in memory controllers.

Soft-fail on data from port- card	Unstriper sees one time error, probability of automatic hardware based data recovery is high	None	
Soft-fail between agg/memory controller dest_id bus	At least two unstripers see either a routeword mismatch, a state with a high number of routeword mismatches, or data parity errors	Queue resynch	
soft-fail between agg/memory controller data bus	Unstriper sees one time error, probability of automatic hardware based data recovery is high	None	
soft-fail between memory controller/separator channel ID bus	At least two unstripers see either a routeword mismatch, a state with a high number of mismatches, or data parity errors	Queue resynch	Tokens get out of synch. May see error of FIFO overflow in the separator, depending on traffic pattern. Need congestion on the fabric for a port to have the FIFO overflow become possible. May also see excess tokens in memory controller.
soft-fail between memory controller/separator data bus for RW data	Packet boundaries from one separator port are lost. Unstriper will show a large number of errors for all traffic from the affected aggregator output.	Queue Resynch	Inherent that no self-stabilize in occurs w/o queue resynch.
soft-fail between memory controller/separator data bus for packet data	Single port sees one-time error.	None	
soft-fail on token bus from separator to memory controller	Mismatches from fabric due to differences in separator scheduling.	Queue Resynch	
soft-fail internal to fabric chips	Unstriper sees different traffic from fabric than other fabrics	Reset	Queue Resynch may fix the problem, reset is necessary for restoring state.
aggregator never sees back plane idle to synchronize to rw bus	Aggregator never sets flag indicating it has seen back plane sync	Replace faulty hardware.	same as below
aggregator never sees system synch	Aggregator never sets flag indicating it has seen back plane sync	Replace Faulty hardware	Locating fault requires see in if only this board is having problems (backplane sync receiver) or if multiple boards are reporting problems (lost both sync signals on the back plane). Error isolation in 40G switch requires looking at the state of the secondary synch pulse generator
memory controller does not see synch from agg		Retry resynch or if permanent replace faulty hardware.	
separator does not see synch from mem_cont	Separator never gets initial synch	replace faulty hardware	
unstriper does not see back	Unstriper never gets back plane	replace faulty hardware	

plane idle	synch		
fabric chips not initialized	Chips do not do anything	Initialize the hardware	Fault can be caused by failure of the on-board processor. If soft-fail, watchdog should catch it.
Striper not initialized	Transmit no data on the back plane	Initialize striper	
Unstriper no initialized	All incoming data ignored	Initialize unstriper	
Stripe amount incorrect	Offending data is dropped in striper, interrupt asserted	Correct stripe amount	Detection comes up as a result of a disagreement between the stripe amount and the configuration register for the switch operating mode.
Primary sync pulse TX failure	Synch pulse receiver on all boards will see error on primary and switch to secondary.	Replace board with primary TX	
Secondary sync pulse TX failure	Synch pulse receiver on all boards will see error on secondary.		
Sync pulse receiver failure on one board	If leaving reset, no chips on board get in sync. If during operation, should see a synch error either in an aggregator or an unstriper fed by this block.	Replace board with bad synch pulse receiver	Need to see how wide error is spread to attempt to identify the source.
Board loses single sync pulse internal to the board	None	If any FIFOs overflow in aggregator or unstriper, queue resynch	
Hard failure on sync pulse distribution to a single chip on a fabric	May see FIFO overflow/under flow in fabric chip or see synch failure from the down stream chip. Additionally, if data is corrupted, the unstriper will report data corruption from the associated fabric.	Replace	
Hard failure on sync pulse distribution to a single chip on a port card	unstriper-May see what looks like a single fabric mismatch due to one fabric going out of synch before the others.	Reset port card	same as below.
soft failure on sync pulse distribution to a single chip on a port card	None	If no FIFO overflow, none. If FIFO overflow, need to reset board(s) with FIFO overflow.	Striper missing synch pulse could overflow a FIFO on every fabric. Recovery would need to be done serially and switch could be effectively down by this error. Only way to ensure all fabrics do the same thing is to ensure that data path has the same delay as the synch path since the writes occur at different logical times. An unstriper missing would affect the output port mapped to the striper and would require a

			port card reset to recover.
soft failure on sync pulse distribution to multiple chips on a fabric	unknown	Reset the fabric	
soft failure on sync pulse distribution to multiple chips on a port card	Same as single-failure case	Same as single-failure	Same as single-failure.

The chipset implements certain functions which are described here. Most of the functions mentioned here have support in multiple ASICs, so documenting them on an ASIC by ASIC basis does not give a clear understanding of the full scope of the functions required.

The switch chipset is architected to work with packets up to 64K + 6 bytes long. On the ingress side of the switch, there are busses which are shared between multiple ports. For most packets, they are transmitted without any break from the start of packet to end of packet. However, this approach can lead to large delay variations for delay sensitive traffic. To allow delay sensitive traffic and long traffic to coexist on the same switch fabric, the concept of long packets is introduced. Basically long packets allow chunks of data to be sent to the queueing location, built up at the queueing location on a source basis and then added into the queue all at once when the end of the long packet is transferred. The definition of a long packet is based on the number of bits on each fabric. The following table gives the size of long packets for different switch sizes.

TABLE 18: Long Packet sizes

Switch Size	Packet Size (bytes)
40	900
80	1800
120	2700
160	3600
240	5400
480	9600

If the switch is running in an environment where Ethernet MTU is maintained throughout the network, long packets will not be seen in a switch greater than 40G in size.

A wide cache-line shared memory technique is used to store cells/packets in the port/priority queues. The shared memory is 8K entries x 200-bit wide running at 125MHz. Each memory controller ASIC yields 25Gbps memory bandwidth. The aggregator #9 (control port) generates at most 4 streams of OC-48 traffic. The enqueue and dequeue speed for different switch configurations is shown in the following table. Note that a 2x speedup can be achieved for all switch configurations except the 480Gswitch. Up to 234,057 cells can be stored in the 480G switch. The shared memory stores cells/packets continuously so that there is virtually no fragmentation and bandwidth waste in the shared memory.

For the short packets/cells, memory utilization can be close to 100%. For the long packets, the memory block before the start of a long packet can be almost completely wasted. The minimum length for a long packet is 3 cache lines, giving an effective utilization of memory close to 75% since 1 out of 4 memory cache lines can be wasted.

TABLE 19: Shared Memory (1,638,400 bits) in Each Memory Controller

Switches	Enqueue Speed	Dequeue Speed	Speedup Ratio	Cell Length	Number of Cells
40G	4.3Gbps	20.7Gbps	4.8	39+1 bits	40,960
80G	4.7Gbps	20.3Gbps	4.3	21+1 bits	74,472
120G	5.0Gbps	20Gbps	4	15+1 bits	102,400
160G	5.3Gbps	19.7Gbps	3.7	12+1 bits	126,030
240G	7Gbps	18Gbps	2.6	9+1 bits	163,840
480G	9.4Gbps	15.6Gbps	1.7	6 +1 bits	234,057

There exists up to 200 queues in the shared memory. They are per-destination and priority based. All cells/packets which have the same output priority and blade/channel ID are stored in the same queue. Cells are always dequeued from the head of the list and enqueued into the tail of the queue. Each cell/packet consists of a portion of the egress route word, a packet length, and variable-length packet data. Cell and packets are stored continuously, i.e., the memory controller itself does not recognize the boundaries of cells/packets for the unicast connections. The packet length is stored for MC packets. There is a limitation of 4K packets (or cells) in each of the MC queues.

The multicast port mask memory 64Kx16-bit is used to store the destination port mask for the multicast connections, one entry (or multiple entries) per multicast VC. The port masks of the head multicast connections indicated by the multicast DestID FIFOs are stored internally for the scheduling reference. The port mask memory is retrieved when the port mask of head connection is cleaned and a new head connection is provided.

Two configurations of port mask memory are supported:

a. 8K port connections, for a 240 G switch

b. 4K connections, for a 480 G switch.

Dequeue performance is restricted by several factors: 1) Padding injected by the aggregator ASICs; 2) Left alignment entries inserted in the memory controllers; 3) Memory controller output bus fragmentation caused by the multicast connections; 4) Token bus latency between the separators and the memory controllers; 5) Separator output bus padding; and 6) Unstriper output bus fragmentation. A 480G switch is used as an example to analyze the worst-case performance since it has most padding, overhead, and congested traffic.

The aggregator ASICs have to pad a packet (including 36-bit route word, variable-length packet length field and datagram) to multiples of 12 since there are 12 memory controllers in one fabric. The shortest packet each memory controller received is 7-bit long since a packet can be as short as 84-bit long. The effective datagram is 3 bits. One entry will be left aligned for every 16 200-bit memory entries. The left aligned entry can be as short as 1-bit long. The worst-case datagram dequeue efficiency per output port of a memory controller is:

*(10-bit (dout_me bus width) * (3/7) (datagram length in a shortest packet) * (15/16) (left-aligned overhead)) * 250MHz (output bus speed) * 12 (number of memory controllers) /24 (number of output ports per separator) = 502Mbps*

The best-case output data bus bandwidth per separator channel is 2-bit * 250MHz, i.e., 500Mbps. In other words, The worst-case dequeue bandwidth of a memory controller is bigger than the best-case output bandwidth of a separator port. 2x speedup can

be achieved through the twice wide output bus of the separators. One sync cycle will be fired on the output bus of the separator every 128 cycles.

The output bus of the unstriper ASIC is 64-bit wide at 100MHz. It can only carry one packet per cycle. In the worst-case, up to 56 bits are wasted per packet for an OC48 port.

APS stands for a Automatic Protection Switching, which is a SONET redundancy standard. To support APS feature in the switch, two output ports on two different port cards send roughly the same traffic. The memory controllers maintain one set of queues for an APS port and send duplicate data to both output ports.

To support data duplication in the memory controller ASIC, each one of 192 unicast queues has a programmable APS bit. If the APS bit is set to one, a packet is dequeued to both output ports. If the APS bit is set to zero for a port, the unicast queue operates at the normal mode. If a port is configured as an APS slave, then it will read from the queues of the APS master port. For OC48 ports, the APS port is always on the same OC48 port on the adjacent port card.

Port mirroring is similar to the APS except that any port can pair with any port. Only one pair of port mirroring ports are supported. A 16-bit port mirror register is used to identify the master and slave port involved in the port mirror operation. All ports are compared to the master portion (bit 15:8) of the register when dequeuing. Port mirror can be disabled. Note that a port can either have APS enabled or port mirroring enable, not both. The value of the port mirror register can be changed on-fly by the shadow registers.

The shared memory queues in the memory controllers among the fabrics might be out of sync (i.e., same queues among different memory controller ASICs have different depths) due to clock drifts or a newly inserted fabric. It is important to bring the fabric queues to the valid and sync states from any arbitrary states. It is also desirable not to drop cells for any recovery mechanism.

A resync cell is broadcast to all fabrics (new and existing) to enter the resync state. Fabrics will attempt to drain all of the traffic received before the resynch cell before queue resynch ends, but no traffic received after the resynch cell is drained until queue resynch ends. A queue resynch ends when one of two events happens:

1. A timer expires.
2. The amount of new traffic (traffic received after the resynch cell) exceeds a threshold.

At the end of queue resynch, all memory controllers will flush any left-over old traffic (traffic received before the queue resynch cell). The freeing operation is fast enough to guarantee that all memory controllers can fill all of memory no matter when the resynch state was entered.

Queue resynch impacts all 3 fabric ASICs. The aggregators must ensure that the FIFOs drain identically after a queue resynch cell. The memory controllers implement the queueing and dropping. The separators need to handle memory controllers dropping traffic and resetting the length parsing state machines when this happens. For details on support of queue resynch in individual ASICs, refer to the chip ADSs.

Multicast connections are enqueued into one of 4 priority queues based on the 2-bit priority number. They are stored cache-line based like the way unicast connections do. Connection numbers and lengths are stored into one of 4 1K-entry per-priority connection FIFO. Multicast packets are subject to be dropped if the destined connection FIFO is full. In other words, at most 1K multicast packets can be stored simultaneously for each priority.

The 64Kx16-bit port mask memory will limit the number of multicast connections supported to 64K, 32K, 16K, 16K, 8K, and 4K for the 40G, 80G, 120G, 160G, 240G, and 480G switch, respectively.

For the dequeue side, multicast connections have independent 32 tokens per port, each worth up 50-bit data or a complete packet. The head connection and its port mask of a higher priority queue is read out from the connection FIFO and the port mask memory every cycle (125MHz). A complete packet (or 50 bits if the packet is longer than 50 bits) is isolated from the 200-bit multicast cache line based on the length field of the head connection. The head packet is sent to all its destination ports. The 8 queue drainers transmit the packet to the separators when there are non-zero multicast tokens are available for the ports. Next head connection will be processed only when the current head packet is sent out to all its ports.

For the worst case analysis, use the 480G switch as an example where the shortest packet is 7 bit long. Every 8ns cycle only one connection can be handled (bottlenecked by the connection FIFO and port mask memory). If the multicast only goes to 1 port, the effective dequeue throughput for the multicast connection is 875Mbps out of available 15Gbps shared memory dequeue bandwidth, i.e., 6%. In other words, the multicast performance is severely damaged by the bottlenecks existing in the connection FIFO, port

mask memory, and head-of-line blocking. The throughput for the 480G switch is 480*7*n/80=n*42G where n is number of copies a multicast connection destined. In the worst case where n=1, the multicast throughput is about 9% available switch capacity. If the average multicast connections make 11 copies, the switch can achieve 480G throughput.

The longer a packet is (for the 240G switch or smaller configurations), the more ports a multicast connection destined, the dequeue performance becomes better significantly. Multicast performance do not intervene the dequeue speedup for unicast connections since the latter has their own tokens and two types of connections share the dout_me bus alternatively in a strict round-robin fashion, i.e., the multicast connections do not block unicast ones.

There are 192 unicast queues, 4 multicast queues, and 4 control port queues. 4 multicast queues are per priority based and can broadcast to any subset of 192 output ports and the 4 control ports.

There are up to 196 destination channels (192 blade channels and 4 control ports) for the 480G switch. Each destination has a one-to-one mapped unicast queue. 4 multicast queues can broadcast to any subsets of 192 regular ports indicated by the per-connection based port mask entry. An OC-192 port uses one out of 4 queue locations. Other three queues are unused. All 8-bit fabric queue ID field on the DestID bus is used to identify one of 196 ports. 2-bit priority field is unused.

For the 240G switch, Up to 100 destination channels exist (96 blade channels and 4 control ports). 96 unicast destination queues have 2 priority queues each. 4 multicast queues can

broadcast to any subsets of 96 ports indicated by the per -connection based port mask entry. An OC-192 port uses one out of 4 queue locations. Other three queues are unused. Lower 7-bit queue ID is used to identify one of 100 ports and lower 1-bit of priority field is used to identify one of two priority queues in each port. Other queue ID bit and priority bit is unused.

For the 160G switch, Up to 68 destination channels exist (64 blade channels and 4 control ports). 64 unicast destination queues have 2 priority queues each. There are 68 unused queues 4 multicast queues can broadcast to any subsets of 68 ports indicated by the per -connection based port mask entry. An OC-192 port uses one out of 4 queue locations. Other three queues are unused. Lower 7-bit queue ID is used to identify one of 100 ports and lower 1-bit of priority field is used to identify one of two priority queues in each port. Other queue ID bit and priority bit is unused.

For the 120G or smaller switch, Up to 52 destination channels exist (48 blade channels and 4 control ports). 48 unicast destination queues have 4 priority queues each. 4 multicast queues can broadcast to any subsets of 48 ports indicated by the per -connection based port mask entry. An OC-192 port uses one out of 4 queue locations. Other three queues are unused. Lower 6-bit queue ID is used to identify one of 52 ports and 2-bit priority field is used to identify one of 4 priority queues in each port. Other queue ID bits are unused.

Queue structure can be changed on fly through the fabric resync cell where the number of priority per port field is used to indicate how many priority queues each port has.

The stripper ASIC resides on the network blade. It has following features:

- Support packet/cell interfaces. Can accept up to 3 GB/sec of sustained traffic (3.2 GB/sec in bursts) of cells, frames, or a mix of cell and frame traffic.
- Generates fabric routeword for all fabrics in the switch
- Calculates data for the parity fabric and adds checksum to the end of each packet.
- Support switch configuration: 40G,80G,120G,160G,240G, and 480G
- Generates appropriate signals to interface directly to the transmit side of the Gbit transceivers.

The Striper takes BIB cell/packet format from the ingress port ASIC. For the ATM interface, the ASX cell format is accepted from the Vortex ASIC of the Poseidon chipset at 2.5Gbps for the channelized blade. It consists of 4-byte route word, 4-byte ATM cell header (without HEC byte), and 48-byte payload. 36-bit the switch route word can be generated based on the ASX route word provided by the Vortex ASIC.

The Striper ASIC consists of three major blocks: the switch route word generator, the switch payload & checksum generator, and the switch parity generator.

The switch payload generator forwards 4-byte ATM cell head, 48-byte ATM cell payload and 2-byte checksum to up to 12 switch fabrics and 1 spare fabric. The cell bus is 2x 12-bit wide running at 125MHz.

The Striper ASIC duplicates the packet/cell and transmits various fragments to the fabrics. 12 data output buses of the striper ASICs are connected to the data input buses of the aggregator ASICs on the fabrics as follows:

Figure 14 shows strip ASIC architecture.

TABLE 20: Data bus connectivity of the Striper ASIC of blade #1

Data bus (DOUT_ST_1_ ch_bu)	40G (1 fabric)	80G (2 fabrics)	120G (3 fabrics)	160G (4 fabrics)	240G (6 fabrics)	480G (12 fabrics)
Bus #1	DIN_AG_1_1_ch_1 =cell[11:0]	DIN_AG_1_1_ch_1 [5:0]=cell[11:6]	DIN_AG_1_1_ch_1 [3:0]=cell[11:8]	DIN_AG_1_1_ch_1 [2:0]=cell[11:9]	DIN_AG_1_1_ch_1 [1:0]=cell[11:10]	DIN_AG_1_1_ch_1 [0]=cell[11]
Bus #2	n/a	DIN_AG_2_1_ch_1 [5:0]=cell[5:0]	DIN_AG_2_1_ch_1 [3:0]=cell[7:4]	DIN_AG_2_1_ch_1 [2:0]=cell[8:6]	DIN_AG_2_1_ch_1 [1:0]=cell[9:8]	DIN_AG_2_1_ch_1 [0]=cell[10]
Bus #3	n/a	n/a	DIN_AG_3_1_ch_1 =cell[3:0]	DIN_AG_3_1_ch_1 [2:0]=cell[5:3]	DIN_AG_3_1_ch_1 [1:0]=cell[7:6]	DIN_AG_3_1_ch_1 [0]=cell[9]
Bus #4	n/a	n/a	n/a	DIN_AG_4_1_ch_1 [2:0]=cell[2:0]	DIN_AG_4_1_ch_1 [1:0]=cell[5:4]	DIN_AG_4_1_ch_1 [0]=cell[8]
Bus #5	n/a	n/a	n/a	n/a	DIN_AG_5_1_ch_1 =cell[3:2]	DIN_AG_5_1_ch_1 [0]=cell[7]
Bus #6	n/a	n/a	n/a	n/a	DIN_AG_6_1_ch_1 =cell[1:0]	DIN_AG_6_1_ch_1 [0]=cell[6]
Bus #7	n/a	n/a	n/a	n/a	n/a	DIN_AG_7_1_ch_1 =cell[5]
Bus #8	n/a	n/a	n/a	n/a	n/a	DIN_AG_8_1_ch_1 =cell[4]
Bus #9	n/a	n/a	n/a	n/a	n/a	DIN_AG_9_1_ch_1 =cell[3]
Bus #10	n/a	n/a	n/a	n/a	n/a	DIN_AG_10_1_ch_ 1=cell[2]
Bus #11	n/a	n/a	n/a	n/a	n/a	DIN_AG_11_1_ch_ 1=cell[1]
Bus #12	n/a	n/a	n/a	n/a	n/a	DIN_AG_12_1_ch_ 1=cell[0]
Spare Fabric Bus	DIN_AG_sp_1_ch_ 1= parity[11:0]	DIN_AG_sp_1_ch_ 1[5:0]= parity[5:0]	DIN_AG_sp_1_ch_ 1[3:0]= parity[3:0]	DIN_AG_sp_1_ch_ 1[2:0]= parity[2:0]	DIN_AG_sp_1_ch_ 1[1:0]= parity[1:0]	DIN_AG_sp_1_ch_ 1[0]= parity[0]

The striper ASICs on blade #1 is connected with aggregator ASIC #1 of all switch fabrics. The striper ASICs on blade #2 is connected with aggregator ASIC #2 of all switch fabrics. The striper ASICs on blade #4 is connected with aggregator ASIC #4 of all switch fabrics. The striper ASICs on blade #5 to #8 are connected with aggregator ASIC #5 to #8 of all switch fabrics, respectively. The striper ASICs on blade #41 to #48 are connected with aggregator ASIC #5 to #8 of all switch fabrics, respectively. In other words, blade number moduled by 8 is the aggregator ASIC number which a striper ASIC is connected to.

The parity bits are sent to the spare fabric. The purpose of the spare fabric is to provide fault tolerance ability to the switch, i.e., in case one of the switch fabrics failed, the spare

fabric recovers the lost part of the cell. This is achieved through a parity bit generator on the striper ASIC. For one fabric configuration, the 12-bit cell payload is duplicated to the spare fabric; for 2-fabric configuration, 6-bit parity bits are generated as follows:

parity bit(1:6) = cell bit(1:6) exclusive-OR cell bit(7:12);

For 3-fabric configuration, 4-bit parity bits are generated as follows:

parity bit(1:4) = cell bit(1:4) exclusive-OR cell bit(5:8) exclusive-OR(9-12);

The route word generator regenerates the switch route word and sends up to 12+1 1-bit 250MHz route word buses for fabric 1,2,3,., 12 and the spare fabric.

The aggregator ASIC resides on the switch fabric as shown in the following figure. Each 40G switch fabric has 8+1 aggregator ASICs. It aggregates 6x4 separate cell streams and route words into a single 12G stream from up to 6 blades and 4 channels. All input signals from the network blades are 250MHz point-to-point HSTL. It outputs a single cell stream that is multiplexed with cell payload and route words to 12 memory controllers. The ASIC has following features:

- 12Gbps Data and route word input from up to 6 network blades and 4 channels
- Route word separation and aggregation
- Output 12G data and route word to 12 memory controller ASICs

- HSTL interface with the memory controller, receiver interface for the backplane gigabit transceivers.

Figure 15 shows aggregator ASIC architecture.

The aggregator ASIC supports 40G, 80G, 120G, 160G, 240G, and 480G switch configuration without backplane change. The backplane connectivity (DIN_AG buses) of a pair of aggregator ASICs is shown as follows:

TABLE 21: DIN_AG bus connectivity of aggregator ASIC #1 and #5 of switch fabric #1

DIN_AG_1_1_ch_bu DIN_AG_1_5_ch_bu	40G (1 fabric)	80G (2 fabrics)	120G (3 fabrics)	160G (4 fabrics)	240G (6 fabrics)	480G (12 fabrics)
DIN_AG_1_1_ch_1	DOUT_ST_1_ch_1=cell[11:0]	DOUT_ST_1_ch_1[5:0]=cell[11:6]	DOUT_ST_1_ch_1[3:0]=cell[11:8]	DOUT_ST_1_ch_1[2:0]=cell[11:9]	DOUT_ST_1_ch_1[1:0]=cell[11:10]	DOUT_ST_1_ch_1[0]=cell[11]
DIN_AG_1_5_ch_1[5: 0]	n/a	DOUT_ST_5_ch_1[5:0]=cell[11:6]	DOUT_ST_5_ch_1[3:0]=cell[11:8]	DOUT_ST_5_ch_1[2:0]=cell[11:9]	DOUT_ST_5_ch_1[1:0]=cell[11:10]	DOUT_ST_5_ch_1[0]=cell[11]
DIN_AG_1_1_ch_2	n/a	n/a	DOUT_ST_9_ch_1[3:0]=cell[11:8]	DOUT_ST_9_ch_1[2:0]=cell[11:9]	DOUT_ST_9_ch_1[1:0]=cell[11:10]	DOUT_ST_9_ch_1[0]=cell[11]
DIN_AG_1_5_ch_2[2: 0]	n/a	n/a	n/a	DOUT_ST_13_ch_1[2:0]=cell[11:9]	DOUT_ST_13_ch_1[1:0]=cell[11:10]	DOUT_ST_13_ch_1[0]=cell[11]
DIN_AG_1_1_ch_3	n/a	n/a	n/a	n/a	DOUT_ST_17_ch_1[1:0]=cell[11:10]	DOUT_ST_17_ch_1[0]=cell[11]
DIN_AG_1_5_ch_3	n/a	n/a	n/a	n/a	DOUT_ST_21_ch_1[1:0]=cell[11:10]	DOUT_ST_21_ch_1[0]=cell[11]
DIN_AG_1_1_ch_4	n/a	n/a	n/a	n/a	n/a	DOUT_ST_25_ch_1[0]=cell[11]
DIN_AG_1_5_ch_4	n/a	n/a	n/a	n/a	n/a	DOUT_ST_29_ch_1[0]=cell[11]
DIN_AG_1_1_ch_5	n/a	n/a	n/a	n/a	n/a	DOUT_ST_33_ch_1[0]=cell[11]
DIN_AG_1_5_ch_5	n/a	n/a	n/a	n/a	n/a	DOUT_ST_37_ch_1[0]=cell[11]
DIN_AG_1_1_ch_6	n/a	n/a	n/a	n/a	n/a	DOUT_ST_41_ch_1[0]=cell[11]
DIN_AG_1_5_ch_6	n/a	n/a	n/a	n/a	n/a	DOUT_ST_45_ch_1[0]=cell[11]

The 2 x 6 DIN_AG buses of aggregator ASIC #1 and #5 pair of switch fabric #1 is connected to the 12 x DOUT_ST bus #1 of blade #1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, and 45, respectively. The 2 x 6 DIN_AG buses of aggregator ASIC #2 and #6

pair of switch fabric #1 is connected to the 12 x DOUT_ST bus #1 of blade #2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, and 46, respectively. The 2 x 6 DIN_AG buses of aggregator ASIC #3 and #7 pair of switch fabric #1 is connected to the 12 x DOUT_ST bus #1 of blade #3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, and 47, respectively. The 2 x 6 DIN_AG buses of aggregator ASIC #4 and #8 pair of switch fabric #1 is connected to the 12 x DOUT_ST bus #1 of blade #4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, and 48, respectively.

Likewise, the 2 x 6 DIN_AG buses of aggregator ASIC #1 and #5 pair of switch fabric #2 is connected to the 12 x DOUT_ST bus #2 of blade #1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, and 45, respectively. The 2 x 6 DIN_AG buses of aggregator ASIC #1 and #5 pair of switch fabric #12 is connected to the 12 x DOUT_ST bus #12 of blade #1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, and 45, respectively, for the 480G switch configuration.

The above connectivity is repeated 4 times for the channelized blades.

For the 40G, 80G, 120G, 160G, 240G, and 480G configuration, each blade channel sends 12 x 36-bit cell payload and 36- bit route word, 6 x 36-bit payload and 36-bit route word, 4 x 36-bit payload and 36-bit route word, 3 x 36-bit payload and 36-bit route word, 2 x 36-bit payload and 36-bit route word, and 1 x 36-bit payload and 36-bit route word to each switch fabric, respectively. In other words, the whole 12-bit wide cell is transmitted in the same fabric for the 40G switch while only a 1-bit wide (1/12 cell) cell slice is transmitted on each fabric for the 480G switch.

The 60-bit DOUT_AG bus is split onto 12 memory controller ASICs, each receiving 5-bit data and 1-bit clock signal from one aggregator ASIC. The 15-bit DestID bus is broadcast to all 12 memory controllers. Due to the fan out load concern, 3 copies of the signals are maintained, each driving 4 ASIC loads.

Every channel of the aggregator sends up to 12x3x200-bit cell/packet stream to 12 memory controller based on a work conserving round-robin dequeue algorithm, i.e., next source takes over if the current source runs out of eligible cells/packets to send. Strict round-robin algorithm is used among 24 sources. For the 40G switch, only 4 source channels exist. A source is eligible to send a cell/packet whenever a full cell or a full short packet or a 12x3x200-bit segment of a long packet is received.

Each memory controller ASIC receives 9 independent cell streams from 9 aggregator ASICs. There are 9 250MHz DIN_ME_fb_se buses, each consisting of a 5-bit data bus, a 1-bit clock signal, and a 15-bit DestID bus. The 60-bit DOUT_AG data buses of all 9 aggregator ASICs are bit sliced onto 12 memory controllers, each receiving 5-bit data from one DOUT_AG bus. Every memory controller gets a separate non-sharing clock signal (named clk1 to clk12) from each DOUT_AG bus to reduce the load of the clock pin while 3 memory controllers share a set of DestID bus from the DOUT_AG bus. The 9 DIN_ME_fb_se buses of memory controller #1 are connected to the DOUT_AG buses of 9 aggregators as follows:

- DIN_ME_fb_1_1_data = DOUT_AG_fb_1_data[48,36,24,12,0]
- DIN_ME_fb_1_1_dest = DOUT_AG_fb_1_dest1
- DIN_ME_fb_1_1_clk = DOUT_AG_fb_1_clk1
- DIN_ME_fb_1_2_data = DOUT_AG_fb_2_data[48,36,24,12,0]

- DIN_ME_fb_1_2_dest = DOUT_AG_fb_2_dest1
- DIN_ME_fb_1_2_clk = DOUT_AG_fb_2_clk1
- DIN_ME_fb_1_3_data = DOUT_AG_fb_3_data[48,36,24,12,0]
- DIN_ME_fb_1_3_dest = DOUT_AG_fb_3_dest1
- DIN_ME_fb_1_3_clk = DOUT_AG_fb_3_clk1
- DIN_ME_fb_1_4_data = DOUT_AG_fb_4_data[48,36,24,12,0]
- DIN_ME_fb_1_4_dest = DOUT_AG_fb_4_dest1
- DIN_ME_fb_1_4_clk = DOUT_AG_fb_4_clk1
- DIN_ME_fb_1_5_data = DOUT_AG_fb_5_data[48,36,24,12,0]
- DIN_ME_fb_1_5_dest = DOUT_AG_fb_5_dest1
- DIN_ME_fb_1_5_clk = DOUT_AG_fb_5_clk1
- DIN_ME_fb_1_6_data = DOUT_AG_fb_6_data[48,36,24,12,0]
- DIN_ME_fb_1_6_dest = DOUT_AG_fb_6_dest1
- DIN_ME_fb_1_6_clk = DOUT_AG_fb_6_clk1
- DIN_ME_fb_1_7_data = DOUT_AG_fb_7_data[48,36,24,12,0]
- DIN_ME_fb_1_7_dest = DOUT_AG_fb_7_dest1
- DIN_ME_fb_1_7_clk = DOUT_AG_fb_7_clk1
- DIN_ME_fb_1_8_data = DOUT_AG_fb_8_data[48,36,24,12,0]
- DIN_ME_fb_1_8_dest = DOUT_AG_fb_8_dest1
- DIN_ME_fb_1_8_clk = DOUT_AG_fb_8_clk1
- DIN_ME_fb_1_9_data = DOUT_AG_fb_9_data[48,36,24,12,0]
- DIN_ME_fb_1_9_dest = DOUT_AG_fb_9_dest1
- DIN_ME_fb_1_9_clk = DOUT_AG_fb_9_clk1

The DIN_ME data buses of memory controller #2 are connected to bit 49,37,25,13, and 1 of the DOUT_AG data buses of 9

aggregators, and so on. The DIN_ME data buses of memory controller #12 are connected to bit 59,47,35,23, and 11 of the DOUT_AG data buses of 9 aggregators.

12 memory controller ASICs aggregate cell/packet streams from 8+1 aggregator ASICs. Then write the cells into one of 200 output queues (e.g., 12 network blades x 4 channelized Poseidon interfaces x 4 priorities for unicast + 4 priorities for multicast + 4 control port queues). The 8-bit destination queue number on the DestID bus is used as the output queue indicator for the unicast connection. The multicast cell is stored into one of 4 priority queues based on the 2-bit priority on the DestID bus. The 16-bit multicast connection number on the DestID bus will be used to lookup the internal port mask memory to find out the destination blade and channels during the dequeue phase.

The memory controllers send out cell/packet traffic from 200 output queues to 8+1 separator ASICs. Dequeuing speed is as twice fast as enqueuing speed to reduce amount of cells buffered on the switch fabric.

- Support both variable-length packet switching and fixed-length cell switching
- 12 ASICs are bit-sliced and function as an integrated shared memory controller
- Support 40G, 80G, 120G, 160G, 240G, and 480G switch configurations
- Enqueue cells/packets from 9 aggregator ASICs
- 2x dequeue speedup to 9 separator ASICs
- On-chip APS support
- 234,057 cells on-chip buffer

- 200 programmable destination queues
- On-chip control port support
- 64K multicast connections, 2^32 unicast connections.
- Per-queue transmit and loss counts

Figure 16 shows memory controller ASIC architecture.

A 8Kx13-bit link list is used to maintain free/used memory entry list pointer. A free entry is requested from the free link list when writing data into the shared memory and the current tail cache line runs out of space. Complete cell/packet will be dropped whenever the free list is empty, i.e., the shared memory is full. A memory entry is free to the free list after the memory word is transmitted to the separator ASICs.

Figure 17 shows wide cache line shared memory architecture.

DIN_ME_fb_se_9 and DOUT_ME_fb_se_9 buses are used to connect to aggregator #9 and separator #9, which communicate with the control port striper and unstriper ASICs only. It has the same DestID and cell format as other 8 buses do. Its cells are enqueued and dequeued in the same way as the regular cells.

There are up to 4 additional control port queues. They have queue ID from 192 to 195. All unicast connections having the control port queue ID as its fabric queue ID is enqueued into the relative control port queue. There are at most 4 OC-12 control ports supported.

Each control port queue has a 13-bit control port register as follows:

TABLE 22: 13-bit Control port queue register

Bit 12:5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8-bit regular port ID	Regular Port enable	Control Port 3 enable	Control Port 2 enable	Control Port 1 enable	Control Port 0 enable

A queue can be multicast to up to 4 physical control ports and one regular queue. When a queue is redirected to the regular queue, that queue must be disabled for the regular queue traffic. Packets are queued in the same way as the regular queues do, i.e., 200-bit cache line based. Left aligned every 16 cache lines. Strict round-robin among 4 queues when a left-alignment entry is transmitted. A queue is routed to 4 control ports and one regular port based on the 5-bit control port enable vector.

Two dequeue algorithms are applied among 4 control port queues:

- a) One control port only talks to one cp queue: Pure round-robin dequeue among 4 non-empty control port queues which have non-zero unicast tokens; one token worth unicast (up to 200-bit) is sent out to dout_me bus for a port;
- b) One control port talks to multicast cp queues: Strict priority among 4 control port queues; queue 192 has highest priority and queue 195 has lowest; switch queues when the end of the packet is seen.

OAM cells are identified by the Fabric queue ID field. If this field of a unicast connection has value 0xFx(h), then it is an OAM cell. All OAM cells can be mapped into one of the 192 blade or

4 control port queues set by a 8-bit programmable register (called OAM cell destination register).

Resync cell (0xFF) or any other special cells with fabric queue ID set to 0xFx are routed to any one of 196 queues based on the OAM cell destination register too.

Per destination minimum and maximum thresholds and counts can be set up to help memory management. 200x2x14-bit thresholds (in unit of 200-bit entry) and 200 x 13-bit running counters (in unit of 200-bit entry) are provided. Two additional per destination transmit and loss counts (32-bit each, in unit of packets) are also maintained. If the running count of a destination is above the relative threshold, new packets are rejected and loss count increments. Whenever dropping, the whole packet is dropped. Otherwise, the transmit count increments. For multicast connections, cells can also be rejected due to the multicast route word FIFO is full. 4 additional FIFO full counts are needed. If a packet is dropped, the whole packet is cleaned from the memory (including the segments of a long packet). The thresholds and current counts are in unit of 200-bit cache lines.

The minimum threshold (13-bit value plus 1-bit enable bit) is used to prevent shared memory starvation, i.e., every queue reserves at least the number of cache lines indicated by the threshold. The maximum threshold (13-bit value plus 1-bit enable bit) is used to prevent any single queue consuming the whole shared memory. These two thresholds cannot be changed unless there are no packets in the queues.

All counters are 32-bit wide. They are reset to zero automatically after reading. Their values stick to 0xFFFFFFFF if

overflowed. It takes 2^32 x 8ns = 32 seconds to overflow a counter in the worst case.

The value of any threshold registers can be updated on-fly by a resync cell or a shadow control cell. The content of the 32- bit shadow data register is copied to the location pointed by the shadow address register.

The memory controller can enqueue a single OC-192 data stream from the aggregator ASIC and dequeue a single OC-192 data stream to the separator ASIC instead of 4xOC-48 streams. At the ingress side, the ASIC receives 4 continuous cells/packets/cache lines from the same source channel instead of 4 channels. No special treatment is needed.

At the egress side, the Queue Drainer reads 4 cache lines from the shared memory for one destination after a token command is received for the OC-192 port. The RCD can send up to 4 200-bit cache lines to the separator from the same destination queue. Each OC-192 port has 4 priorities for all switch configurations.

The separator ASICs receive cell/packet streams from 12 memory controllers, separate, and send them up to 48 network blades through the backplanes. The interfaces between the separator and the backplane are 250MHz point-to-point HSTL signals.

Figure 18 shows the Separator ASIC architecture.

- Receive 12 data streams from 12 memory controllers
- Fabric synchronization
- 24-destination (blades and channels) addressing

- Route word separation and aggregation
- 0.25um 3V CMOS technology
- 410 I/O pins
- 140-bit 250MHz input; 240-bit 250MHz output (at most 120 of them switch simultaneously); 30-bit control signals

The separator has twice number of data output pins as that of the aggregator ASIC to support 2X speedup. Similar to those of the striper ASIC, the ASIC supports 40G, 80G, 120G, 160G, 240G, and 480G switch configurations without backplane change.

The separator ASIC performs reverse function of the aggregator ASIC. The ASIC receives 120-bit 250MHz cell/packet stream from one of 8 DOUT_ME_fb_se_bu buses of every memory controller (12 of them). 10-bit blade and channel selection signals are used to select one of 24 destinations inside each separator for up to two cells. For example, the DIN_SP buses of separator ASIC #1 is connected as follows:

- DIN_SP_fb_1_1=DOUT_ME_fb_1_1
- DIN_SP_fb_1_2=DOUT_ME_fb_2_1
- DIN_SP_fb_1_3=DOUT_ME_fb_3_1
- DIN_SP_fb_1_4=DOUT_ME_fb_4_1
- DIN_SP_fb_1_5=DOUT_ME_fb_5_1
- DIN_SP_fb_1_6=DOUT_ME_fb_6_1
- DIN_SP_fb_1_7=DOUT_ME_fb_7_1
- DIN_SP_fb_1_8=DOUT_ME_fb_8_1
- DIN_SP_fb_1_9=DOUT_ME_fb_9_1
- DIN_SP_fb_1_10=DOUT_ME_fb_10_1
- DIN_SP_fb_1_11=DOUT_ME_fb_11_1

- DIN_SP_fb_1_12=DOUT_ME_fb_12_1
- CH_SP_fb_1 = CH_ME_fb_1

When a valid cell/packet (channel ID is in the range of 0-23) is received, the packet type field in the route word is checked first. If it is an ATM cell, no packet length field is followed. The length of cell payload is 36x12/number of fabrics. If it is a packet, the packet length bit immediately followed is used to indicate how long a packet length is. 0=12-bit packet length (including this bit) and 1=24-bit packet length (including this bit). The entire packet/cell is routed to the destination channel indicated by the channel ID. The invalid channel ID (bigger than 24) is used to indicate that the cell/packet is invalid.

The ASIC then separate the route word and the payload onto the route word bus and the data bus of one of 6 blades and 4 destination channels/unstriper ASICs based on the channel ID signals. One 250MHz 24-bit data bus yields 6Gbps data bandwidth for each channel. Each route word is 2-bit wide running at 250MHz.

The connectivity between the separator ASICs and the Unstriper ASICs are symmetric to those between the aggregator ASICs and the striper ASICs. The only difference is that all data and route word pins have double-width to achieve 2X speedup.

Data received from each destination of each memory controller has a 1-bit valid bit accompanied. There are 24 destination input FIFOs are used to store the 12 pieces of cell/packets from 12 memory controllers for 24 destination blade and channels in each separator, respectively. When all 12 cell segments arrives, the complete cell is sent to the relative output FIFO indicated by the channel ID.

Like the striper ASIC, a 3-bit sequence number counter is maintained for the backplane synchronization. It increments every 36 250MHz cycles. When a cell is sent to the unstriper ASICs via the backplane, the current counter is attached into the sequence number field in the 36-bit route word.

The sequence number counter is reset by the global resynchronization logic.

The unstriper ASIC takes 6Gbps traffic from up to 12+1 switch fabrics. It then unstripes the cell and send it to the egress netmod ASIC at 5Gbps or lower speed.

- Receive 6Gbps route word and data from up to 12+1 fabrics at 250MHz for OC48 or combine 4 chips to support 20 Gbps routeword and data from up to 12+1 fabrics for OC192c
- Error check data transport throughout the switch, detect corrupted data and perform data recovery
- Reconstructs cells/packets from the individual switch fabrics.
- Send 64-bit 100MHz data to the egress port ASIC for OC48, 256 bit for OC192c
- Supports both UC and MC connection context for fabric data.

Figure 19 shows the unstriper ASIC Architecture.

The unstriper ASIC receives cells from up to 12+1 fabrics, each running at 250MHz. It uses the following steps to reconstruct good data.

1. All incoming routewords are compared. If any one routeword disagrees, that data lane is flagged as being in error. If more than one routeword disagrees, the data is dropped.

2. All valid input lanes are put through reconstruction logic which will attempt to build n+1 candidate output data streams for an N fabric switch. Any data lane which is not valid will invalidate any data lane which uses that data.

3. All valid reconstruction lanes will check the CRC of the received data and one passing output is selected.

The striper remaps the separate routeword and data buses to a combined outgoing routeword +data bus.

The following will detail the steps which happen at power up from an architectural perspective. Note that when expanding switch capacity, the additional fabrics must be brought on-line before any new port cards are brought on-line.

Fabric Initialization

1. Port cards (unstripers) are initialized to only look at current fabric capacity and ignore other fabric inputs.
2. Fabric is inserted, asserts its board present signal. Stripers start sending routewords to the new fabrics, though they are ignored at this point.
3. Board is reset, MCP starts to boot the board. Before proceeding to the next step, the MCP/SCP establish communication via the e-net network.
4. If the board is fabric 0 or the parity fabric, the sync pulse transmitter is initialized. (Actually sync pulse transmitter

can be initialized on all fabrics, but it is only connected to BP signals if it is fabric 0 or the parity fabric.)

5. MP initializes sync registers in the aggregator, memory controller, and separator, then initializes the registers in the sync pulse receiver. The sync pulse receiver starts to look for a valid sync pulse. The last sync setup is the sync pulse receiver, so that all receivers on the chips are ready for the sync pulse from the sync pulse receiver. The fabric chips run chip-chip sync on the next backplane sync pulse. The MP should check to make sure the fabric has synchronized. If sync has not been achieved, reset the fabric chips and re execute step 4.

6. SCP tells MP the current switch capacity window to use. This is actually going to correspond to the current switch capacity (does not count the capacity of the new fabric if switch capacity is being expanded).

7. MP initializes the backplane transceiver networks with the current switch capacity (both send and receive) and initializes all registers except the aggregator input enables. Any values used for configurable options (which ports are OC48/OC192, memory thresholds, etc) need to be communicated and initialized at this point. Certain registers are initialized based on the switch board slot, which needs to be known at this point. From a software perspective, the biggest register set which must be done is to update the port mask table in the memory controllers to match the port mask table from another switch fabric.

8. Aggregator input enables are set for the current switch capacity. This will start enqueueing traffic on this switch board. The aggregators will need to see a bus idle followed by an increment in the transmit sequence number before starting to actually receive data.

9. SCP sends a queue resync cell. On cell return, fabric queues

are now synchronized. However, no valid data is being enqueued in the new fabric(s) and the fabric outputs are being ignored.

10. All unstripers must be configured to start utilizing the new fabric. Since queues have been resynchronized, the fabric dequeuing should be synchronized and no errors should be seen. If errors are seen, clear them, return to step 8.

11. After all unstripers have been updated, SCP tells all port card MCPs to update stripe amount inside each of the striper ASICs. The change in striper configuration will start the switch utilizing the additional capacity.

12. After all stripe amounts are updated and traffic from the previous stripe amount drained from the switch, then the switch capacity needs to be updated. The only fixed time bound way of ensure traffic from the previous stripe amount is flushed is to execute a queue resync. If not all traffic has been flushed from the system with the previous stripe amount, the switch will drop this traffic at the unstripers (since there is no synchronization of the update at the separators, the drop cannot be performed there).

Before a port card is brought on-line, any necessary switch fabrics must be brought on-line first. As per the switch standard convention, port card installation happens in order.

1a. The starting state has sufficient switch capacity to support the new port card. Aggregators are currently configured to ignore the input from any new board.

1b. Port card is inserted and asserts its board present signal. Port card sees sync pattern received from the fabrics.

2. The sync pulse receiver is initialized. The port card starts looking for a valid sync pulse on the backplane.

4. Striper transmitter is set up for the appropriate number of destination fabrics and the Gbit network control is initialized. Before the GBit networks are initialized, the fabrics cannot count on seeing idle data from the new port card. At this point, the port card can communicate its type (OC48/OC192) to the fabrics.

5a. Fabrics configure the port card type and enable the input from the port card.

5b. Striper/unstriper are now initialized, along with the other chips on the board. Some enable in the inbound data path should be disabled. The BIB input enable in the striper can be used or some other board specific input enable.

6. After both 5a and 5b have been completed, the port card can enable its input side and start sending data to the fabrics. Note that in general, further software configuration will need to be done after this point (such as setting up inbound lookup entries). The completion of 5a is necessary to ensure the fabric queues do not go out of sync.

7. First data from the port card is striped to all fabrics.

8. When a port card is removed from the system, not very much needs to happen from a hardware perspective. Before the port card goes away, it transmits a packet abort which will cause any incomplete packets in the egress side to the dropped. Traffic will be drained from the memory queues which correspond to the affected output ports.

9. To remove a port card from the switch logically, software should disable the striper output bus.

Fabric deactivation is similar to fabric activation in reverse. The steps include:

1. Switch capacity is being removed. If port cards are present in the switch which are paired with the fabric capacity which is about to be removed, those must first be deactivated.

2. Program the remaining stripers in the system to stripe data to one less stripe amount than the current configuration. This will stop sending real data to the fabric about to be decommissioned.

3. Send a queue resynch. This will flush out any traffic at the last stripe amount.

4. Program the unstripers to start ignoring the data from the fabric which is about to be removed.

5. The fabric can now be physically removed from the system, or logically removed from the system by disabling its inputs and outputs.

The reason for the queue resynch step is not because the switch is out of sync. The unstriper will treat the receipt of traffic which is striped to more fabrics than physically present in the switch as an error and increment error counts. The queue resynch ensures that the error counts on the unstripers will not increment unnecessarily.

1. Flush out traffic from the port to be converted over to APS. Initialize anything in the separator as required for the new output port combination.

2. Write to the APS enable bit using the shadow register in every memory controller for the output port being affected. The main port for APS is not affected. Either a higher or lower number port can be the primary port and the backup port. APS is always enabled on the backup port.

3. Send either a queue resync cell or a shadow control cell to all memory controllers.

4. Memory controllers start to dequeue after the next left-aligned cache boundary (if the previous transfer for this port was left-aligned, it will be remembered).

Note that in all this process, the queue number was never switched. The switch will not support a seamless port swap due to APS activate/deactivate. (In other words, APS can be turned on port 0, which will cause port 0 to mirror port 16. However, APS cannot be turned off on port 16 since it is not on. Traffic is only being changed for the port where APS is added.)

The following words have reasonably specific meanings in the vocabulary of the switch. Many are mentioned elsewhere, but this is an attempt to bring them together in one place with definitions.

TABLE 23:

Word	Meaning
APS	Automatic Protection Switching. A sonet/sdh standard for implementing redundancy on physical links. For the switch, APS is used to also recover from any detected port card failures.
Backplane synch	A generic term referring either to the general process the the switch boards use to account for varying transport delays between boards and clock drift or to the logic which implements the TX/RX functionality required for the the switch ASICs to account for varying transport delays and clock drifts.
BIB	The switch input bus. The bus which is used to pass data to the striper(s). See also BOB
Blade	Another term used for a port card. References to blades should have been eliminated from this document, but some may persist.
BOB	The switch output bus. The output bus from the striper which connects to the egress memory controller. See also

	BIB.
Egress Routeword	This is the routeword which is supplied to the chip after the unstriper. From an internal chipset perspective, the egress routeword is treated as data. See also fabric routeword.
Fabric Routeword	Routeword used by the fabric to determine the output queue. This routeword is not passed outside the unstriper. A significant portion of this routeword is blown away in the fabrics.
Freeze	Having logic maintain its values during lock-down cycles.
Lock-down	Period of time where the fabric effectively stops performing any work to compensate for clock drift. If the backplane synchronization logic determines that a fabric is 8 clock cycles fast, the fabric will lock down for 8 clocks.
Queue Resynch	A queue resynch is a series of steps executed to ensure that the logical state of all fabric queues for all ports is identical at one logical point in time. Queue resynch is not tied to backplane resynch (including lock- down) in any fashion, except that a lock-down can occur during a queue resynch.
SIB	Striped input bus. A largely obsolete term used to describe the output bus from the striper and input bus to the aggregator.
SOB	One of two meanings. The first is striped output bus, which is the output bus of the fabric and the input bus of the agg. See also SIB. The second meaning is a generic term used to describe engineers who left Marconi to form/work for a start-up after starting the switch design.
Sync	Depends heavily on context. Related terms are queue resynch, lock-down, freeze, and backplane sync.
Wacking	The implicit bit steering which occurs in the OC192 ingress stage since data is bit interleaved among stripers. This bit steering is reversed by the aggregators.

The Aggregator Receive Synchronizer's function is to maintain logical cell/packet ordering across all fabrics. Cells/packets arriving at more than one fabric from different port cards need to be processed in the same logical order across all fabrics. If cell/packet logical ordering is not maintained, then cells/packets coming out of fabrics will have stripes of a particular cell/packet not match up and will not be able to be re-assembled by the Unstriper.

Logical cell/packet ordering needs to be maintained across the following conditions:

- Transport delay variances between one source and multiple destinations
- Clock drift across transmitters and receivers

- Insertion and removal of port cards and fabrics
- Port card errors such as no sync, no lock-downs, too fast/too slow, routeword parity errors
- Gigabit transceiver errors such as loss-of-lock, data errors
- Non-synchronized updates to Gigabit network
- OC192c data streams(aggregating 4 channels to make up one OC192c stream)

The switch uses a system of transmit and receive counters. The counters allow all components in the system to logically align themselves. The Master Sequence Generator implements these two counters that will count continuously from `0' to `3' and will increment every x 125 MHz clock cycles where, x is the counter tick length as programmed by software. x is currently calculated to be 250 cycles. This is based on analysis done in the Backplane Synchronization ADS. The relationship between the transmit and receive counters can be seen in Figure 20. One counter will be used by the transmit synchronizers in the Striper and Separator ASICs and the other counter will be used in the receive synchronizers in the Aggregator and Unstriper ASICs. The receive counter will be a delayed version of the transmit counter. The amount of delay is programmed by software in the Sync Pulse Receive Delay register. This register determines the number of clock cycles that the receive counter waits before incrementing its own counter relative to the transmit counter. This register should always be non-zero since the transmitter will have no delay and the receiver needs to be delayed with respect to the transmitter. The Sync Pulse Receive Delay has been estimated to be 150 cycles. The delay is approximated equal to the worst case transport delay between transmitter and receiver plus worst case transport delay variance of the sync pulse. The delay also takes into account worst case fast and slow transmitters and receivers.

The Sync Pulse Period is defined as the number of cycles between sync pulses. It is extended slightly by about 10 cycles in order for it to appear late in the `0' window of each ASIC's sequence count. This is done to ensure that every ASIC will appear to be running too fast even if they are actually running slow relative to the clock that generated the sync pulse. If this was not done, the sync pulse could appear in the `3' window and the ASIC would consider itself to be slow. There would be no way for it to catch up. Each transmitter and receiver will calculate the difference between when the sync pulse arrives and when its own counter transitions from `3' to `0'. This difference is the number of cycles that it is fast and is referred to as the lock-down amount(z in figure). Once a transmitter determines it should lock-down for z cycles, it will finish sending valid data during its `0' window and then lock-down z cycles. During the lock-down period, no valid or idle data is sent. Instead, a special lock-down K character is transmitted which will be recognized by the receiver. The receiver will not write the lock-down characters into its input FIFOs. This will ensure that the input FIFOs can't overflow. Since the sequence counter does not advance for the amount of lock-down, it is effectively resetting itself to the sync pulse. It is equivalent of having the sync pulse appear at the start of the `0' count window since the transition to a count of `1' occurs precisely one tick length after the sync pulse arrives. When the next sync pulse arrives, if clock frequencies are constant, then the sync pulse should appear in the `0' count window and the calculated lock-down amount will be the same as the previous calculation. This allows the system to always expect the sync pulse arrival in the `0' count window even if the clocks generating the sequence counter are too fast or too slow.

The Receive Synchronizer block will use the sequence counter to determine when to accept data from input byte sync FIFOs. Once a sync character is read, pops from the FIFOs will only occur once the sequence counter transitions from "0" to "1" and immediately following an arrival of a sync pulse. The read decision is only made once every sync pulse arrival and only at the "0" to "1" transition of the receive sequence counter. The sequence counter is also used during fabric resync in order to communicate a fabric resync to all channels in all aggregators during a sequence count transition. Fabric resync cells will be transmitted at the beginning of a sequence tick window and are prefixed by a special character indicating a resync cell. The receive synchronizers in the Aggregators will resynchronize all data going to the memory controllers on the next sequence count transition once the resync character has been received.

A block diagram of the receive Synchronizer can be seen in Figure 21. The Receive Synchronizer consists of 24 Byte-sync FIFOs, a Crossbar and 6 Bus Synchronizers. There is one byte sync FIFO per gigabit receiver. Each byte sync FIFO will accept data from each gigabit receiver independent of the mode of the switch. The byte sync FIFO depth is about 256 words deep. This depth is based on a derivation found in the Backplane Synchronizer ADS. The Crossbar will handle the assignment of the appropriate input byte lanes to the correct channels. Each Bus Synchronizer will consist of four Channel FIFOs and one Bus Controller. The Bus Controller can handle 4 separate OC48 channels or one OC192c stream. The channel FIFO is about 18 words deep. The depth is based on the number of words to read a 36-bit routeword. The whole routeword is read and then presented to the rest of the Aggregator in one cycle since it needs to be stored before the data of the packet as it is constructed and sent to the memory controller.

Multiple gigabit receivers make up a 24-bit data bus and 2-bit routeword bus for one channel of an Aggregator. Each gigabit receiver can handle up to 8 bits. Due to varying transport delays that can exist between receivers, bytes from different receivers that belong to the same word can be skewed from each other. For example, the 24-bit data bus and 2-bit routeword bus for one channel of an aggregator will have 4 receivers that make up the bus. The synchronization logic will align all 4 bytes for the 26-bit bus and will pass this byte aligned word to the rest of the Aggregator. In order to align the bytes, the Striper will need to send a special alignment byte to each receiver. A special K character can be utilized from the gigabit transceivers. The K character will be encoded in the data bits on the Gigabit transmitter and will be detected on the Gigabit receiver.

The receive synchronizer in the Aggregator will consist of 24 FIFOs where there is one FIFO per Gigabit Receiver. These FIFOs will handle both byte alignment and the backplane synchronization. It is assumed that the Gigabit Receivers will be able to distinguish between valid, idle, sync and lock-down cycles and will indicate these various cycles to the Aggregator by using 3 control signals.

On startup, the FIFOs will be empty and each Write State Machine(WSM) will wait until a sync character is seen on its input. From this point on, every cycle will be pushed except for lock-down cycles from the fabric. When the fabric is locking down, the Stripers will send special lock-down characters. This is done to avoid overflowing the sync FIFOs in case the write side clock is faster than the read side clock. While particular types of words are being pushed, the word type will also be written to the FIFO so it can be distinguished on the read side.

The WSM is also looking for a special fabric resync cell K character that will indicate that a fabric queue resync cell will immediately follow. If a resync cell is detected, a resync signal is passed along to Bus Controller. The Bus Controller will then tell other Aggregators on the fabric to resync their queues at the next transition of the sequence counter. Fabric queue resync is described in more detail later.

Gigabit receivers are not dedicated to particular input channels, but instead shared between various channels. Each byte sync FIFO works independently of the switch mode and each input lane needs to be steered to the correct channel FIFO. For instance in 40 mode, 26 bits of data and routeword are required for Bus 1, channel A and therefore 4 byte lanes are required to be steered to each channel of Bus 1. In 80/120 mode, only 8 bits of data and 2 bits of routeword are required and therefore two bytes will suffice. In 480 mode, only 4 bits are required per channel and one byte lane will suffice. As switch capacity increases, less and less byte lanes will be required for a particular channel. For all switch modes, the routeword bits for a particular channel will always come from the same byte lane. As the byte lanes get reduced from 4 to 1 byte lanes, there will always be one common byte lane used to carry the routeword data lines. The crossbar will take in 24 lanes consisting of 8 bits of data and 3 bits of control along with other control signals to communicate with the Bus Control logic. It will then forward all these signals to the appropriate channels. The Crossbar will also accept control data from the Bus Controller and forward signals such as read requests and FIFO flush signals to the appropriate input byte sync FIFOs. Each crossbar mapping between input byte lanes and channels is bi-directional.

The Bus Controller consists of three state machines. The state machines control the read side of the byte sync FIFOs, the

write side of the channel FIFOs and the read side of the Channel FIFOs. On the read side of the Byte FIFOs, pops will not commence until a sync pulse has arrived and the receive sequence counter has transitioned from "0" to "1". A signal will be provided from the sequence generator block that indicates a "0" to "1" transition at precisely this moment(*sync_event*). At this time, the Bus Controller issues a read to the Crossbar for the particular channel. The Crossbar then forwards the read signal to the appropriate byte sync FIFOs based on the mode of the switch. The Crossbar then forwards all data and control from these byte sync FIFOs back to the Bus Controller for this channel. The Bus Controller checks the data types to make sure that the first word in the appropriate byte sync FIFOs are a sync character. If the first word of any of the appropriate byte lanes for this channel is not a sync character, then a sync error will be flagged, appropriate byte sync FIFOs will be flushed and the synchronization process will be re-initiated. If the first word is a sync character, then pops will continue. In OC48 mode, this process will be performed independently for each channel. OC192c support is discussed later on.

Once data starts being read from byte sync FIFOs, the Bus Controller will ignore data until it finds the first idle word. Once an idle word has been found, it can now start looking for the SOP indication in the routeword when the next non-idle word is read. The rest of the routeword is processed and made available to the rest of the Aggregator. If the stop bit in the routeword indicates that the packet is continuing, then data will be continuously made available to the Aggregator until a stop indication is read. Note that even though a SOP is seen, it does not mean that this segment is the first segment of a packet. It can be any segment of a packet. Even though the segment may not be

the first one of a packet, it is allowed to go through the switch and will be dropped later on.

When a sync character is read, a counter is initialized. The counter counts each read from the byte sync FIFOs. The Bus Controller will expect to see a sync character every sync pulse period(about 22,000 cycles). If a sync character is read too early or too late, then a sync error is flagged, data is dropped at the precise logical cycle of where a sync character is expected. A packet that is being processed at the theoretical logical cycle for sync will be terminated and inputs will be disabled until re-enabled by S/W. For example, if after the first sync character, the next sync character occurs at cycle 19,000, and then a sync error is flagged. Data is not dropped until 22,000 reads have been performed. Also, if after the first sync character, the next sync character is not received at all after 22,000 cycles, then a sync error is flagged and data is dropped at this precise logical cycle. If a sync character is received precisely 22,000 cycles after the last one, then reads from the byte sync FIFOs are stopped until the receive sequence counter transitions from `0' to `1'. Waiting for the `0' to `1' transition will ensure that all fabrics are receiving the same stripe of a packet on the same logical cycle.

For OC192c, 4 input channels need to be concatenated into one OC192c stream. In this mode, the Bus Controller will control all 4 channel FIFOs and the appropriate byte sync FIFOs. Data type checking will be performed across 4 times as many byte lanes as in the OC48 case. When it is time to read byte sync FIFOs, the Bus Controller will control 4 read control lines to the Crossbar. The Crossbar will initiate reads across all appropriate byte sync FIFOs that are required for OC192c and will present data back to the Bus Controller. The Bus Controller will check data types and will look for SOP indications. The SOP indication and stop bits will only be

found in the Routeword for channel A. The Bus Controller will write all 4 channel FIFOs at the same time when writing data and will present the complete OC192c Routeword in one cycle to the rest of the Aggregator. The functions of the Bus Controller will be identical for OC48 and OC192c except that all 4 channel FIFOs will be controlled when in OC192c mode.

Special cases can be broken down into the following categories:

Port card insertion

1. Port card removal
2. Port card errors including:
   - A. No sync character
   - B. Port card not locking down
   - C. Routeword parity errors
   - D. Garbage data
   - E. Port card sending data too fast or too slow
3. Fabric Queue resync
4. Non-synchronized updates to Gigabit network

When a port card is inserted, the port card present signal will be asserted and sent to each fabric. Not until S/W enables the particular inputs and the Aggregator sees the port card present signal, will the Aggregator be ready to accept data from the new port card. Once enabled, the Aggregator will go through the process of looking for sync characters on individual byte lanes associated with the new port card. It is assumed that the port card will not send any data until it has been configured only after the fabrics have been initialized. Once the port cards are enabled, they will start sending sync characters periodically at every global sync pulse arrival. It is important that all the appropriate fabrics see the sync character from the particular port

card since some fabrics will be initialized later than others. After sync characters have been received, all data will be written on each cycle excluding lock-down characters.

When a port card is about to be removed, the enable switch on the port card will be turned off. This will signal the port card to finish sending valid packets and then send idles. The port card will send a packet abort k character to indicate that no more valid packets will be sent immediately following the last valid packet. It is assumed that when the port card is actually removed, it will have already sent the packet abort k character. This is critical for the fabrics to keep their queues in sync. It is important that each Aggregator on each fabric that handles the particular port card stops forwarding data to the memory controllers at precisely the same logical cycle. The WSM will stop writing data into the byte sync FIFOs once the packet abort character is seen. The Bus Controller will terminate the packet once the packet abort character is read out of the byte sync FIFOs.

Case A: No sync/early sync/late sync from port card.
Solution: The Synchronizer will look for a sync at precisely the same logical cycle each time. This will occur every sync pulse period that is approximately 22,000 125MHz cycles. If the sync character is not present at the head of the byte sync FIFOs when 22,000 cycles have been read since the last sync character, a sync error will be flagged and data will be dropped the cycle where the sync character should have been. All fabrics need to drop data at precisely the same logical cycle for this particular input lane. Inputs for this particular channel will be turned off and the byte sync FIFOs used for this channel will be flushed. S/W will turn off the offending Striper. Inputs will be ignored until S/W enables these inputs again. If a sync character arrives too early, then data should be dropped at precisely the cycle where the early

sync was read. Other Aggregators will make the same drop decision if this error is common to all fabrics. If the sync character arrives too late or not at all, then the drop decision will be made where the sync character was expected. The sync character is expected to arrive every 22,000 cycles after the last sync.

Case B: Port card not locking down.
Solution: If the port card does not lock-down, it will then send more than the ideal number of valid and idle cycles between sync characters. This will be caught by the same logic that checks for sync characters in the correct logical cycles. Data will be dropped the same way as in the case where no sync came from the port card.

Case C: Routeword parity errors.
Solution: If a parity error is detected for a particular routeword, the packet will be terminated at the bad segment and a parity error will be flagged. Data will be dropped after this terminated segment is forwarded to the rest of the Aggregator and FIFOs for this particular channel will be flushed. Inputs will be disabled until re-enabled by S/W.

Case D: Garbage data from port card while all fabrics already in sync.
Solution: If the data is unrecognizable by the gigabit receivers, errors will be formed and provided to the Aggregator by the gigabit receivers. At the point of error, data being written into byte sync FIFOs will be flagged to be in error. If the Bus Controller sees that the particular byte lane in error is not used for the Routeword bits, then the error will be flagged but the data will be passed on to downstream logic. This is considered to be a soft failure since queues will still be able to stay in sync. If the Bus Controller sees that the particular byte lane in error is used

for the Routeword bits, then the packet will be terminated and then dropped once the erred word is read from the byte sync FIFO. The input will be disabled, a gigabit receiver error will be flagged to S/W and byte sync and channel FIFOs associated with this channel will be flushed. This is considered to be a hard failure. If the failure occurs only for one fabric, then other fabrics can still be used to re-assemble the packets. S/W will have to queue resync the bad fabric. If this error occurs across multiple fabrics, not much can be done to avoid fabric queues from becoming corrupted. S/W will then have to queue resync all fabrics.

Case E: Port card sending data too fast or too slow. It is possible that the port card is sending the correct number of valid cycles between sync characters but is not locking down enough or locking down too much during each lock-down period. Byte sync FIFOs can eventually overflow or underflow respectively. If more than one fabric have FIFOs that overflow or underflow and data is dropped at different logical cycles for the same source, then fabric queues can become out of sync.

Solution: This is considered a hard failure since it should not occur if the hardware is working correctly. The only way to possibly prevent this is to flag an error if the FIFOs reach an almost full or almost empty threshold. This is a warning sign that something is wrong. S/W will then turn off the offending port card. Data will continue to be written to and read from the byte sync FIFOs as if nothing is wrong. If the port card can be turned off and idles be sent before byte sync FIFOs overflow, then there will be no dropped data and fabric queues will stay in sync. If FIFOs overflow or underflow for a particular channel, then a FIFO overflow/underflow error will be flagged. The packet being processed by the synchronizer at the time of error will be terminated. All data will be dropped from this point on. Inputs

for this channel will be disabled until re-enabled by S/W. FIFOs for this channel will be flushed.

Fabric queue resync is performed in order to resynchronize memory controller queues. It is important that all fabrics are processing the stripe of the same cell or packet at precisely the same logical cycle and that all fabrics are acting together as one logical fabric. Fabric queue resync starts at the Stripers. The Striper will receive a queue resync cell from the control port. The striper will decode the queue resync cell and will back up traffic until the next sequence counter tick is reached. At this point, it will send a fabric queue resync K character immediately followed by the queue resync cell. At the fabric, the WSM in the receive synchronizer will receive the queue resync K character and notify the Bus Controller in the receive synchronizer that a queue resync cell is in the input FIFO and that the queue resync event should occur at the next transition of the receive sequence counter. The Bus Controller will then indicate to other Aggregators on the fabric that a resync cell event will take place at the next transition of the sequence counter. The indication is asserted about 10 cycles before the receive sequence counter transitions. This is done to allow enough time for other Aggregators to see this assertion before their respective receive sequence counters transition also. Once the sequence count transition occurs, the Aggregators will signal to the memory controllers that a queue resync event has occurred and that this event delimits old and new data. All data sent before the sync event is considered old data and all data sent after the sync event is considered new data. The memory controllers synchronize their buffers accordingly. The resync cell is eventually sent through the switch as a regular cell and returned to the control port.

There can be times when the gigabit network is changing its operating mode and the switch is changing from a 40/80 to an 80/120 mode for example. There is no guarantee that Gigabit Receivers will be driven by Gigabit Transmitters during this time period. Aggregators that expect good data from certain Gigabit Receivers may not get good data. If the switch is increasing its mode, then a previously unused FIFO will now be used. If this FIFO has garbage data on its inputs, then syncs will not be received and this FIFO will not be synced until the gigabit network is stable. Once the Gigabit network is stable, idles and sync characters will be transmitted by the port cards and the FIFOs will have enough time to sync up. If the switch is decreasing its mode, then previously used FIFOs will now be unused. The Aggregator will know the new switch capacity and will eventually ignore these channel FIFOs.

The Unstriper needs to provide back-pressure to the Separators when internal FIFOs in the Unstriper become near full. Each Separator will expect 24 separate back-pressure signals coming from all the port card channels it is connected to. The back-pressure signal is considered to be asynchronous to all ASICs. It is required that all relevant Separators receive back-pressure from a particular channel in the Unstriper at precisely the same logical cycle. This is done by having the Unstripers assert the back-pressure signal when their receive sequence counter transitions. It is assumed that the Unstriper's receive sequence counter is a delayed version of the Stripers transmit sequence counter. Since the tick length is 250 cycles and the receive counter is delayed by 150 cycle relative to the transmit counter, there exists 100 cycles of margin to transport the back-pressure signal from the Unstriper to the Separator. The Separator needs about 10 cycles before the transition of its sequence counter to sample the back-pressure signal. This will give the Separator enough time to provide back-

pressure to the memory controller before the counter transitions. This places a maximum requirement on the propagation delay of the back-pressure signal. The following requirements hold true:

Back-pressure propagation delay < counter tick length - receive sync pulse delay - setup time of Separator' sample point

Back-pressure propagation delay < 250 - 150 - 10
Back-pressure propagation delay < 90 cycles @ 125 MHz or 720 ns

Assuming worst-case conditions, the expected worst-case propagation delay would be:

Back-pressure propagation delay = (Unstriper to Striper delay) + (Striper to Aggregator delay) + Aggregator to Separator Delay
Back-pressure propagation delay = 5 cycles (chip and board delay) + (5+62 cycles)(chip and port card to fabric delay of 500 ns) + 5 cycles (chip and board delay)
Back-pressure propagation delay = 77 cycles < 90 cycles

As can be seen from this estimate, the maximum back-pressure propagation delay requirement is met.

Assuming all the relevant Separators receive the back-pressure signal before the transition to the next sequence count, then it can be synchronized to the next transition of the transmit sequence counter. This will allow all relevant Separators to stop sending valid data at precisely the same logical cycle for one complete counter tick interval. This is true since it is assumed that when the transmit sequence counter transitions, the data that the Separators are sending are companion fragments of the same packet. If back-pressure is sampled again before the next counter transition, then data will be stopped for another counter

tick interval. This mechanism implies that back-pressure can only be generated on a counter tick length granularity.

Since there is no direct path from Unstriper to Separator, the back-pressure signals need to be re-routed from the Unstriper, to the Striper, to the Aggregator and finally to the Separator. In order to do this, each Unstriper needs to send the back-pressure signal to the corresponding Striper on that port card. The Striper will then forward the back-pressure signal through the backplane gigabit transceivers onto the Aggregator. The Aggregator will forward up to 24 separate back-pressure signals to one Separator corresponding to 6 buses with 4 channels per bus. The back-pressure signal will always use bit 0 of the gigabit transceivers. The receive synchronizer block in the Aggregator will forward the correct back-pressure signal for the appropriate bus and channel to the Separator. Since the gigabit receivers are not dedicated to any particular bus and channel, the synchronizer needs to select the correct gigabit receiver based on the switch configuration just like it does for regular data. Once this is done, bit 0 of the gigabit receiver is forwarded on as the back-pressure signal. Note that bit 0 is also used for receiving k characters and can change when sending a k character. In order to avoid mistakenly interpreting bit 0 of a k character as a valid back-pressure signal, the synchronizer will only sample the back-pressure bit when valid data is received from the gigabit receiver. In the case where a k character is received, the synchronizer will hold the back-pressure signal at its current value. There is still a case where the Striper can be sending back-to-back idle characters since there is nothing to send. If the Striper needs to change the value of the back-pressure signal in this case, then it will send one of two k characters that change the back-pressure value. The two k characters that will be used are a set and clear of the back-pressure signal. If the synchronizer receives a

back-pressure set or clear character, it will set or clear the back-pressure signal respectively. If any other k character is received, the current back-pressure signal is retained. If valid data is received, bit 0 of the appropriate gigabit receiver is sampled as the back-pressure signal.

Although the invention has been described in detail in the foregoing embodiments for the purpose of illustration, it is to be understood that such detail is solely for that purpose and that variations can be made therein by those skilled in the art without departing from the spirit and scope of the invention except as it may be described by the following claims.